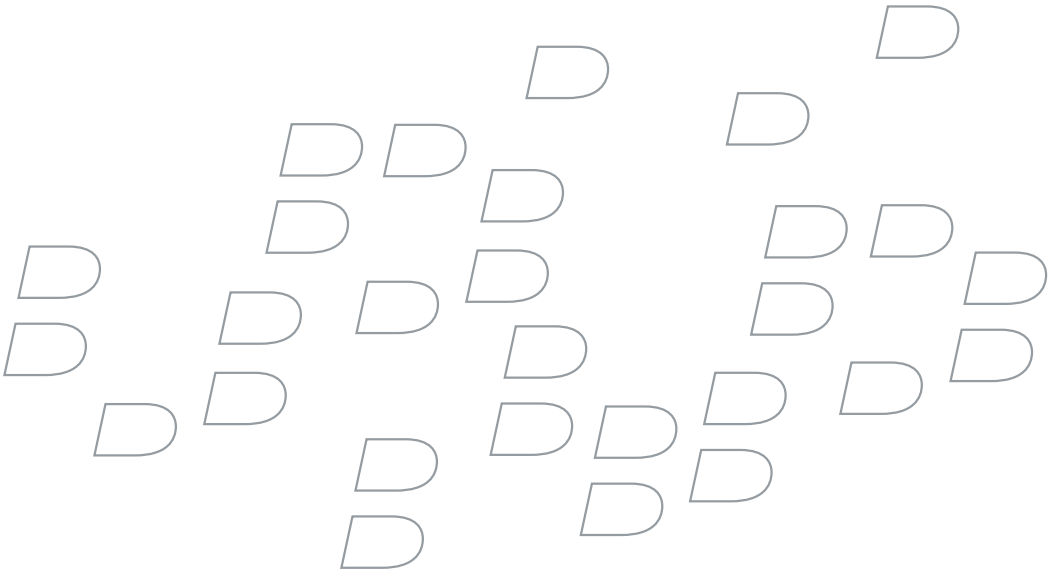

BlackBerry Java Development Environment

Version 4.2.0

Development Guide

BlackBerry Java Development Environment Version 4.2.0 Development Guide

Last modified: 11 February 2008

Part number: 9461716

At the time of publication, this documentation is based on the BlackBerry Java Development Environment Version 4.2.0.

Send us your comments on product documentation: https://www.blackberry.com/DocsFeedback.

©2008 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images, and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, BlackBerry, "Always On, Always Connected" and the "envelope in motion" symbol are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

The Bluetooth word mark and logos are owned by the Bluetooth SIG, Inc. and any use of such marks by Research In Motion Limited is under license. Java and JavaScript are either registered trademarks or trademarks of Sun Microsystems, Inc. in the United States or other countries. Microsoft, ActiveX, Internet Explorer, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. iCal is either a registered trademark or trademark of Apple Computer, Inc. in the U.S and other countries. vCard is either a registered trademark or trademark of the Internet Mail Consortium. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The BlackBerry device and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit www.rim.com/patents.shtml for a list of RIM (as hereinafter defined) patents.

This document is provided "as is" and Research In Motion Limited and its affiliated companies ("RIM") assume no responsibility for any typographical, technical or other inaccuracies in this document. In order to protect RIM proprietary and confidential information and/or trade secrets, this document may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third party sources of information, hardware or software, products or services and/or third party web sites (collectively the "Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the Third Party Information or the third party in any way. Installation and use of Third Party Information with RIM's products and services may require one or more patent, trademark or copyright licenses in order to avoid infringement of the intellectual property rights of others. Any dealings with Third Party Information, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third party. You are solely responsible for determining whether such third party licenses are required and are responsible for acquiring any such licenses relating to Third Party Information. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use Third Party Information until all such applicable licenses have been acquired by you or on your behalf. Your use of Third Party Information shall be governed by and subject to you agreeing to the terms of the Third Party Information licenses. Any Third Party Information that is provided with RIM's products and services is provided "as is". RIM makes

no representation, warranty or guarantee whatsoever in relation to the Third Party Information and RIM assumes no liability whatsoever in relation to the Third Party Information even if RIM has been advised of the possibility of such damages or can anticipate such damages.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House, 36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Published in Canada

Contents

1 Creating UIs 13
Elements of a BlackBerry device UI 13
Screens 13
UI components 14
Create a screen 15
Adding UI components to a screen 15
Create UI components 15
Creating custom UI components 17
Create a custom field 18
Code sample: Creating custom buttons 22
Create custom context menus 26
Code sample: Creating a custom context menu 27
Create custom layout managers 29
Code sample: Creating a custom layout manager 30
Create custom lists 33
Code sample: Creating a custom list 34
Adding menu items to BlackBerry applications 36
Create a menu item 36
Register a menu item 36
Code sample: Creating a new menu item in a BlackBerry application 37
Arrange UI components 38
Define a layout manager 40
Set field focus and navigation 40
Listen for field focus changes 40
Respond to UI events 40
Listen for field property changes 42
Manage foreground events 43
Manage drawing areas 43

2 Using graphics and multimedia 45
Using images 45
Use raw images 45
Use encoded images 46

Code sample: Using a raw image to recreate an encoded image ... 47
Drawing and rendering images ... 48
Position an image ... 48
Draw an image in color ... 48
Using audio ... 51
Start the media player from the BlackBerry Browser ... 51
Create a media player ... 51
Play media ... 54
Listen for media player events ... 54
Using rich media ... 55
Playing rich media content ... 55
Code sample: Retrieving and displaying a rich media file ... 56
Listen for rich media events ... 57
Code sample: Managing rich media content download events ... 58
Code sample: Implementing a listener to download rich media content ... 59
Create a custom connector for rich media connections ... 62
Code sample: Implementing a custom connector ... 62

3 Storing data ... 65
Use BlackBerry persistent storage ... 65
BlackBerry persistent storage ... 65
Manage persistent data ... 66
Code sample: Saving user name and password information ... 67
Manage custom objects ... 69
Code sample: Storing and viewing restaurant information ... 71
Use the MIDP record store ... 75

4 Managing data ... 77
Data synchronization ... 77
Types of data synchronization ... 77
Backing up and restoring data ... 78
Add support for backing up data over the wireless network ... 78
Access a SyncCollection ... 80
Notify the system when a SyncCollection changes ... 81
Using SyncObjects ... 82
Code sample: Using a SyncCollection to back up data over the wireless network ... 82
Add support for backing up data with the BlackBerry Desktop Software ... 88

Activate synchronization when the BlackBerry device starts 89
Code sample: Letting the BlackBerry Desktop Software back up and restore application data 89

5 Using smart cards 97
Smart cards 97
Adding support for unsupported smart cards 97
Creating a smart card driver 97
Create a smart card object 97
Code sample: Creating a smart card object 98
Activate libMain() on startup 101
Create a session for the smart card driver 101
Code sample: Creating a smart card session 101
Create a CryptoToken for private key operations 106
Code sample: Creating a CryptoToken for private key RSA operations 108
Store the private key file location 112
Code sample: Storing the location of a private key file on the smart card 112

6 Managing memory 115
Invoking a garbage collection operation 115
Reduce the number of objects 115
Managing low memory 115
LMM triggers 115
Use the LMM 116
Free persistent objects 116

7 Creating connections 117
Fetching data using HTTP or TCP sockets 117
Explicitly selecting a gateway 117
Using the BlackBerry Enterprise Server as an intranet gateway 117
Using the wireless service providers Internet gateway 118
Use HTTP connections 118
Use HTTP authentication 122
Use HTTPS connections 125
Use socket connections 125
Datagram connections 126
Use datagram connections 126
Using port connections 128

Use USB or serial port connections 128
Use Bluetooth serial port connections 128

8 Creating notifications 135
Types of notification events 135
Add a new event source 136
Register the event source when the BlackBerry device starts 136
Code sample: 137
Triggering events 139
Respond to deferred events 140
Cancel events 141
Customize system notifications for immediate events 141
Code sample: Creating a custom notification 143

9 Managing applications 147
Application manager 147
Retrieve information about applications 147
Register applications when the BlackBerry device starts 148
Communicate with other applications 148
Determine the services that are available to BlackBerry applications 148
Listen for changes to IT policies 149
Code example: Listening for changes to IT policies 149
Managing code modules 149
Retrieve module information 150
Retrieve an array of handles for existing modules on a BlackBerry device 150
Create code modules 150
Runtime store 151
Share runtime objects 151

10 Using the messages application 153
Create new messages 153
Work with a message 155
Open a message 155
Send a message 156
Reply to a message 157
Forward a message 158
Work with folders 158

Working with attachments 160
Create an attachment handler 160
Retrieve attachments 160
Send a message with an attachment 161

11 Using PIM applications 163
Using the calendar 163
Start the calendar from your application 163
Use the calendar 164
Code sample: Creating new recurring appointments 167
Using the address book 170
Open the address book from your application 170
Use contacts 170
Code sample: Displaying a screen that lets BlackBerry device users add new contacts 175
Using tasks 178
Start the task application from your application 178
Use tasks 179
Code sample: 182

12 Using the phone application 185
Start the phone application from your application 185
Use phone call functionality 185
Add DTMF tones to the send queue 186
Listen for phone events 187
Access and use call logs 187
Code sample: Calculating the time that a participant spends on the phone 188

13 Using the BlackBerry Browser 191
Display content in the BlackBerry Browser 191
Display content in a BlackBerry Browser field 192
Code sample: Using the BlackBerry Browser 194

14 Using location information 199
Types of location information 199
Using BlackBerry Maps 199
Location documents 199
Create a location document to display map data 200

Use the BlackBerry Maps application ... 201
Locating BlackBerry devices using GPS information ... 201
Methods for retrieving a GPS location ... 201
Selecting a GPS location provider ... 202
Retrieve BlackBerry device GPS location information ... 204
Requirements for retrieving GPS location information from BlackBerry devices that run on the CDMA network ... 205
Send required PDE data to BlackBerry devices that run on the CDMA network ... 206
Code sample: Recording GPS information for a BlackBerry device ... 206

15 Creating push applications ... 215
Types of push applications ... 215
Types of push requests ... 216
Write a client push application ... 217
Code sample: Listening for data from a web server ... 218
Write a server-side push application ... 222
Work with a server-side push request ... 224
Create a RIM push request ... 225
Create a PAP push request ... 226
Code sample: Pushing data to an application that listens on a BlackBerry device ... 227

16 Localizing applications ... 235
Storing text strings in resource files ... 235
Storing resources for a locale ... 235
Files required for localization ... 236
Add localization support ... 236
Code sample: Storing text strings in separate resources for locales ... 237
Retrieve strings from a resource file ... 240
Code sample: Retrieving strings from a resource file ... 240
Manage resource files for application suites ... 243

17 Testing applications ... 245
Testing applications using the BlackBerry IDE ... 245
Use the BlackBerry device simulator to test synchronizing data with the BlackBerry Desktop Software ... 245
Testing applications using BlackBerry devices ... 246
Connect the BlackBerry IDE to a BlackBerry device ... 246

Debugging applications ... 246
Use breakpoints ... 246
Debug an application in the BlackBerry IDE ... 247
Manage a debugging session ... 248
Locate an error in the source code ... 248
Run an application to the insertion point ... 248
Debug an application on a BlackBerry device ... 248
Step through lines of code in an application ... 250
View statistics to locate memory leaks ... 250
Display objects in memory to locate object leaks ... 251
View local variables ... 252
View variable or expression information ... 252
View static data ... 252
Evaluate (watch) Java expressions ... 253
View threads ... 253
View the data members of a process ... 254
View the call stack ... 254
View event logs ... 254
View classes ... 255
Optimize source code using the BlackBerry IDE profiler tool ... 255
Analyze code coverage ... 257
Start the BlackBerry email simulator ... 259
Working with compiled applications ... 260
Load and remove applications ... 260
View application information ... 261

18 Packaging and distributing applications ... 263
Preverify applications ... 263
Determine if your code requires signatures ... 263
Controlled APIs ... 263
Register to use RIM controlled APIs ... 264
Restricted access to code signatures ... 265
Request code signatures ... 265
Request code signatures using a proxy server ... 266
Request a replacement registration key ... 266
View signature status ... 266
Distributing applications over the wireless network ... 267

Distribute applications 267
Distributing applications with the BlackBerry Desktop Software 269
Create an application loader file 269
Load an application on a specific BlackBerry device 269
Specify optional components 270
Specify supported BlackBerry Device Software 271

A Appendix: The command line compiler 273
Using the command line compiler 273

B Appendix: XML control entity attributes 275
Using XML control entity attributes 275

C Appendix: .alx files 277
Elements in BlackBerry application .alx files 277

D Appendix: BlackBerry application .jad files 281
Properties of BlackBerry application .jad files 281

Acronym list 283

1

Creating UIs

Elements of a BlackBerry device UI
Create a screen
Adding UI components to a screen
Creating custom UI components
Adding menu items to BlackBerry applications
Arrange UI components
Set field focus and navigation
Listen for field focus changes
Respond to UI events
Listen for field property changes
Manage foreground events
Manage drawing areas

Elements of a BlackBerry device UI

Screens

The main structure for a BlackBerry® device UI is the `Screen` object. A BlackBerry device application may display more than one screen at a time, but only one screen in an application is active at one time.

The UI APIs initialize simple `Screen` objects. Once you create a screen, you can add fields and a menu to the screen and display it to the BlackBerry device user by pushing it onto the UI stack. The `menu` object has associated menu items that are runnable objects, which perform a specific task when the BlackBerry device user selects one of the items. For example, menu items may invoke the necessary code to establish a network connection, commit a data object to memory, or close the application. For more sophisticated custom applications, you can customize the BlackBerry device UI and implement new field types, as required. You can also add custom navigation and trackwheel behavior.

The `Screen` class does not implement disambiguation, which is required for complex input methods, such as international keyboards and the BlackBerry 7100 Series. For seamless integration of the different input methods, extend `Field` or one of its subclasses. Do not use `Screen` objects for typing text.

Types of screens

Screen Type	Class	Description
`Default`	`Screen`	Use the `Screen` class to define a manager to lay out UI components on the screen and to define a specific type of screen using the styles that the constants on the `Field` superclass define.
`Standard vertical`	`FullScreen`	By default, a `FullScreen` class contains a single vertical field manager. Use a `FullScreen` class to provide an empty screen that you can add UI components to in a standard vertical layout. For another layout style, such as horizontal or diagonal, use a `Screen` class and add a `Manager` to it.
`BlackBerry style`	`MainScreen`	The `MainScreen` class provides features that are common to standard BlackBerry® applications. Use a `MainScreen` object for the first screen of your application to maintain consistency with other BlackBerry applications. The `MainScreen` class provides the following UI components: • default position of a screen title, with a `SeparatorField` after the title • main scrollable section contained in a `VerticalFieldManager` • default menu with a **Close** menu item • default close action when the BlackBerry device user clicks the **Close** menu item or presses the **Escape** key

How the JVM manages screens

The JVM maintains `Screen` objects in a display stack, which is an ordered set of `Screen` objects. The screen at the top of the stack is the active screen that the BlackBerry® device user sees. When an application displays a screen, it pushes the screen to the top of the stack. When an application closes a screen, it removes the screen off the top of the stack and displays the next screen on the stack, redrawing it as necessary. Each screen can appear only once in the display stack. The JVM throws a runtime exception if a `Screen` that the application pushes to the stack already exists. Applications must remove screens from the display stack when the BlackBerry device user finishes interacting with them so that the application uses memory efficiently. Use only a few modal screens at one time, because each screen uses a separate thread.

UI components

Fields represent all UI components, which are rectangular regions that a `Manager` contains. A field's layout requirements determine the size of the field. Managers provide scrolling for the fields that they contain.

To create a specialized field component (such as a text field that contains multiple elements), create your own custom types by extending the `Field` class or one of its subclasses.

Traditional field	BlackBerry field
Button	`ButtonField`
Check box	`CheckboxField`
Date	`DateField`
Dialog box	`PopupScreen`
Drop-down list	`NumericChoiceField or ObjectChoiceField`
Option	`RadioButtonField`
Text	`RichTextField, BasicEditField, EditField,PasswordEditField, or AutoTextEditField.`
Text label	`LabelField`
List	`ListField`

Create a screen

> Extend the `Screen` class or one of its subclasses, `FullScreen` or `MainScreen`.

Adding UI components to a screen

1. Create an instance of a UI component.

   ```
   CheckboxField myCheckbox = new CheckboxField("First checkbox", true);
   ```

2. Add the UI component to your extension of a screen class.

   ```
   mainScreen.add(myCheckbox);
   ```

Create UI components

To create an instance of a component, you can use more than one constructor. See the *API Reference* for more information about `Field` classes.

Task	Steps

Create a pop-up screen.	1. Create an instance of a subclass of the `Manager` class.
	`Manager manageLayout = new HorizontalFieldManager(VERTICAL_SCROLLBAR);`
	2. Create an instance of a `PopupScreen` using the `Manager` object.
	`PopupScreen popUp = new PopupScreen(manageLayout);`

Add a bitmap.	> Create an instance of a `BitmapField`. `BitmapField myBitmapField = new BitmapField();`
Create a button.	> Create an instance of a `ButtonField` using a style parameter. `ButtonField mySubmitButton = new ButtonField("Submit");`
Create a numeric drop-down list	> To create a drop-down list that contains numbers, create an instance of a `NumericChoiceField`. `NumericChoiceField myNumericChoice = new NumericChoiceField( "Select a number: ", 1, 20, 10);`
Create a numeric drop-down list for a large range of numbers.	> Create an instance of a `GaugeField`.
Create an alphanumeric drop-down list.	> To create a drop-down list that contains objects, create an instance of an `ObjectChoiceField`, providing an object array as a parameter. `String choiceItems[] = {"Option one", "Option two", "Option three"};`
Create a check box.	> Create an instance of a `CheckboxField`. `CheckboxField myCheckbox = new CheckboxField("First checkbox", true);`

Task	Steps

Create an option.	1. Create an instance of a `RadioButtonGroup()`.
	`RadioButtonGroup rbGroup = new RadioButtonGroup();`
	2. Create an instance of a `RadioButtonField` for each option you want to make available to the BlackBerry® device user.
	`RadioButtonField rbField = new RadioButtonField("First field");`
	`RadioButtonField rbField2 = new RadioButtonField("Second field");`
	3. Invoke `RadioButtonGroup.add()` to add the `RadioButtonFields` to the `RadioButonGroup` and make sure the BlackBerry device user can select only one option at a time.
	`rbGroup.add(rbField);`
	`rbGroup.add(rbField2);`

Create a date field.	> Create an instance of a `DateField`, providing the value returned by `System.currentTimeMillis()` as a parameter to return the current time. `DateField dateField = new DateField("Date: ", System.currentTimeMillis(), DateField.DATE_TIME);`
Create a read-only field that you can format using different fonts and styles.	> Create an instance of a `RichTextField`. `RichTextField rich = new RichTextField("RichTextField");`
Create an editable text field that contains no default formatting but accepts filters.	> Create an instance of a `BasicEditField`. `BasicEditField bf = new BasicEditField("BasicEditField: ", "", 10, EditField.FILTER_UPPERCASE);`
Create an editable text field that lets BlackBerry device users access special characters.	> Create an instance of an `EditField`. `EditField edit = new EditField("EditField: ", "", 10, EditField.FILTER_DEFAULT);`
Create a password field.	> Create an instance of a `PasswordEditField`. For example, the following instance uses a constructor that lets you provide a default initial value for the `PasswordEditField`. `PasswordEditField pwd = new PasswordEditField("PasswordEditField: ", "");`
Create an AutoText edit field.	> Create an instance of an `AutoTextEditField`. `AutoTextEditField autoT = new AutoTextEditField("AutoTextEditField: ", "");` Some filters render some AutoText entries ineffective. For example, `FILTER_LOWERCASE` renders an AutoText entry that contains capitalization ineffective.
Create a field that displays a progress bar for the numbers that the BlackBerry device user selects.	> Create an instance of a `GaugeField`. `GaugeField percentGauge = new GaugeField("Percent: ", 1, 100, 29, GaugeField.PERCENT)`
Create a text label.	> Create an instance of a `LableField` to add a text label to a screen. `LabelField title = new LabelField("UI Component Sample", LabelField.ELLIPSIS));`

Task	Steps

Create a field that lets a BlackBerry device user select a range of items in the list.	1. Create the items that you want to display in a `ListField`.
	`String fieldOne = new String("Mark Guo");`
	`String fieldTwo = new String("Amy Krul");`
	2. Create an instance of a `ListField`.
	`ListField myList = new ListField();`
	3. Create an instance of a `ListCallback`.
	`ListCallback myCallback = new ListCallback();`
	4. Set the call back of the `ListField` to be the `ListCallback`.
	`myList.setCallback(myCallback);`
	5. Use the `ListCallBack` object to add items to the `ListField`.
	`myCallback.add(myList, fieldOne);`
	`myCallback.add(myList, fieldTwo);`
	6. Add the `ListField` to the MainScreen.
	`mainScreen.add(myList);`

Create a field that displays a folder or tree relationship between items such as documents or message folders.	A `TreeField` contains parent and child nodes.
	1. To draw a `TreeField`, implement the `TreeFieldCallback` interface.
	2. Specify whether a folder is collapsible by invoking `setExpanded()` on the `TreeField` object.
	`String fieldOne = new String("Main folder");`
	`...`
	`TreeCallback myCallback = new TreeCallback();`
	`TreeField myTree = new TreeField(myCallback, Field.FOCUSABLE);`
	`int node1 = myTree.addChildNode(0, fieldOne);`
	`int node2 = myTree.addChildNode(0, fieldTwo);`
	`int node3 = myTree.addChildNode(node2, fieldThree);`
	`int node4 = myTree.addChildNode(node3, fieldFour);`
	`...`
	`int node10 = myTree.addChildNode(node1, fieldTen);`
	`myTree.setExpanded(node4, false);`
	`...`
	`mainScreen.add(myTree);`
	Your implementation of TreeFieldCallback should add fields to the tree. See "Create custom lists" on page 33 for more information about creating callbacks.
	`private class TreeCallback implements TreeFieldCallback {`
	`public void drawTreeItem(TreeField _tree, Graphics g, int node, int y, int width, int indent) {`
	`String text = (String)_tree.getCookie(node);`
	`g.drawText(text, indent, y);`
	`}`
	`}`

Creating custom UI components

To create custom fields, content menus, layout managers, and lists, use the BlackBerry® APIs.

Create a custom field

Task	Steps

Create a custom field.	You can only add custom context menu items and custom layouts to a custom field.
	> Extend the `Field` class, or one of its subclasses, implementing the `DrawStyle` interface to specify the characteristics of the custom field and turn on drawing styles.
	`public class CustomButtonField extends Field implements DrawStyle {`
	`public static final int RECTANGLE = 1;`
	`public static final int TRIANGLE = 2;`
	`public static final int OCTAGON = 3;`
	`private String _label;`
	`private int _shape;`
	`private Font _font;`
	`private int _labelHeight;`
	`private int _labelWidth;`
	`}`

Define the label, shape, and style of the custom button.	> Implement constructors to define the label, shape, and style of the custom button.
	`public CustomButtonField(String label) {`
	`this(label, RECTANGLE, 0);`
	`}`
	`public CustomButtonField(String label, int shape) {`
	`this(label, shape, 0);`
	`}`
	`public CustomButtonField(String label, long style) {`
	`this(label, RECTANGLE, style);`
	`}`
	`public CustomButtonField(String label, int shape, long style) {`
	`super(style);`
	`_label = label;`
	`_shape = shape;`
	`_font = getFont();`
	`_labelHeight = _font.getHeight();`
	`_labelWidth = font.getWidth();`
	`}`

Task	Steps

Specify the arrangement of the objects in the field.	1. Implement `layout()`. Arrange field data so that you perform the most complex calculations in `layout()` instead of in `paint()`.
	2. Within your implementation, perform the following actions:
	• To calculate the available width and height, invoke `Math.min()` to return the smaller of the specified width and height and the preferred width and height of the field.
	• To set the required dimensions for the field, invoke `setExtent(int, int)`.
	• Recalculate the pixel layout, cached fonts, and locale strings.
	`protected void layout(int width, int height) {`
	`_font = getFont();`
	`_labelHeight = _font.getHeight();`
	`_labelWidth = _font.getAdvance(_label);`
	`width = Math.min( width, getPreferredWidth() );`
	`height = Math.min( height, getPreferredHeight() );`
	`setExtent( width, height );`
	`}`
	The manager of the field invokes `layout()` to determine how the field arranges its contents in the available space.

Define the preferred width of a custom component.	> Implement `getPreferredWidth()`, using the relative dimensions to make sure that the label does not exceed the dimensions of the component.
	`public int getPreferredWidth() {`
	`switch(_shape) {`
	`case TRIANGLE:`
	`if (_labelWidth < _labelHeight) {`
	`return _labelHeight << 2;`
	`} else {`
	`return _labelWidth << 1;`
	`}`
	`case OCTAGON:`
	`if (_labelWidth < _labelHeight) {`
	`return _labelHeight + 4;`
	`} else {`
	`return _labelWidth + 8;`
	`}`
	`case RECTANGLE: default:`
	`return _labelWidth + 8;`
	`}`
	`}`

Task	Steps

Define the preferred height of a custom component.	> Implement `getPreferredHeight()`, using the relative dimensions of the field label to determine the preferred height.
	`public int getPreferredHeight() {`
	`switch(_shape) {`
	`case TRIANGLE:`
	`if (_labelWidth < _labelHeight) {`
	`return _labelHeight << 1;`
	`} else {`
	`return _labelWidth;`
	`}`
	`case RECTANGLE:`
	`return _labelHeight + 4;`
	`case OCTAGON:`
	`return getPreferredWidth();`
	`}`
	`return 0;`
	`}`

Task	Steps

Define the appearance of the custom field.	1. Perform complex calculations in `layout()` instead of in `paint()`.
	2. Implement `paint()`.
	`protected void paint(Graphics graphics) {`
	`int textX, textY, textWidth;`
	`int w = getWidth();`
	`switch(_shape) {`
	`case TRIANGLE:`
	`int h = (w>>1);`
	`int m = (w>>1)-1;`
	`graphics.drawLine(0, h-1, m, 0);`
	`graphics.drawLine(m, 0, w-1, h-1);`
	`graphics.drawLine(0, h-1, w-1, h-1);`
	`textWidth = Math.min(_labelWidth,h);`
	`textX = (w - textWidth) >> 1;`
	`textY = h >> 1;`
	`break;`
	`case OCTAGON:`
	`int x = 5*w/17;`
	`int x2 = w-x-1;`
	`int x3 = w-1;`
	`graphics.drawLine(0, x, 0, x2);`
	`graphics.drawLine(x3, x, x3, x2);`
	`graphics.drawLine(x, 0, x2, 0);`
	`graphics.drawLine(x, x3, x2, x3);`
	`graphics.drawLine(0, x, x, 0);`
	`graphics.drawLine(0, x2, x, x3);`
	`graphics.drawLine(x2, 0, x3, x);`
	`graphics.drawLine(x2, x3, x3, x2);`
	`textWidth = Math.min(_labelWidth, w - 6);`
	`textX = (w-textWidth) >> 1;`
	`textY = (w-_labelHeight) >> 1;`
	`break;`
	`case RECTANGLE: default:`
	`graphics.drawRect(0, 0, w, getHeight());`
	`textX = 4;`
	`textY = 2;`
	`textWidth = w - 6;`
	`break;`
	`}`
	`graphics.drawText(_label, textX, textY, (int)( getStyle() & DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ), textWidth );`
	`}`
	The fields manager invokes `paint()` to redraw the field whenever an area of the field is marked as invalid.

Paint a field only within the visible region.	> Invoke `Graphics.getClippingRect()`.
Manage focus events.	> Use the `Field.FOCUSABLE` style and implement `Field.moveFocus()`.
Change the appearance of the default focus indicator.	> Override `drawFocus()`.

Task	Steps

Add component capabilities.	> Implement the `Field` `set()` and `get()` methods.
	`public String getLabel() {`
	`return _label;`
	`}`
	`public int getShape() {`
	`return _shape;`
	`}`
	`public void setLabel(String label) {`
	`_label = label;`
	`_labelWidth = _font.getAdvance(_label);`
	`updateLayout();`
	`}`
	`public void setShape(int shape) {`
	`_shape = shape;`
	`updateLayout();`
	`}`

Code sample: Creating custom buttons

Example: CustomButtonField.java

```
/**
 * CustomButtonField.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custombuttons;

import net.rim.device.api.ui.*;
import net.rim.device.api.system.*;

/**
 * CustomButtonField  is a class that creates button fields of various
 * shapes. This sample demonstrates how to create custom UI fields.
 */

public class CustomButtonField extends Field implements DrawStyle {
    public static final int RECTANGLE = 1;
    public static final int TRIANGLE = 2;
    public static final int OCTAGON = 3;

    private String _label;
    private int _shape;
    private Font _font;
    private int _labelHeight;
    private int _labelWidth;

    /* Constructs a button with specified label, and the default style and shape. */
    public CustomButtonField(String label) {
        this(label, RECTANGLE, 0);
    }
```

```
/* Constructs a button with specified label and shape, and the default style. */
public CustomButtonField(String label, int shape) {
    this(label, shape, 0);
}

/* Constructs a button with specified label and style, and the default shape. */
public CustomButtonField(String label, long style) {
    this(label, RECTANGLE, style);
}
/* Constructs a button with specified label, shape, and style */
public CustomButtonField(String label, int shape, long style) {
    super(style);
    _label = label;
    _shape = shape;
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);
}

/* Method that draws the focus indicator for this button and
 * inverts the inside region of the shape.
 */
protected void drawFocus(Graphics graphics, boolean on) {
    switch(_shape) {
        case TRIANGLE:
            int w = getWidth();
            int h = w >> 1;
            for (int i=h-1; i>=2; --i) {
                graphics.invert(i, h - i, w - (i << 1), 1);
            }
            break;
        case RECTANGLE:
            graphics.invert(1, 1, getWidth() - 2, getHeight() - 2);
            break;
        case OCTAGON:
            int x3 = getWidth();
            int x = 5 * x3 / 17;
            int x2 = x3 - x;
            x3 = x3 - 1;
            x2 = x2 - 1;
            graphics.invert(1, x, getWidth() - 2, x2 - x + 1);

            for (int i=1; i<x; ++i) {
                graphics.invert(1+i, x-i,
                        getWidth() - ((i+1)<<1), 1);
                graphics.invert(1+i, x2+i,
                        getWidth() - ((i+1)<<1), 1);
            }
            break;
        }
}

/* Returns the label. */
public String getLabel() {
    return _label;
}
/* Returns the shape. */
public int getShape() {
```

```
        return _shape;
    }

    /* Sets the label. */
    public void setLabel(String label) {
        _label = label;
        _labelWidth = _font.getAdvance(_label);

        updateLayout();
    }

    /* Sets the shape. */
    public void setShape(int shape) {
        _shape = shape;
        updateLayout();
    }
    /* Retrieves the preferred width of the button. */
    public int getPreferredWidth() {
        switch(_shape) {
            case TRIANGLE:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight << 2;
                } else {
                    return _labelWidth << 1;
                }
            case OCTAGON:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight + 4;
                } else {
                    return _labelWidth + 8;
                }
            case RECTANGLE: default:
                return _labelWidth + 8;
        }
    }

    /* Retrieves the preferred height of the button. */
    public int getPreferredHeight() {
        switch(_shape) {
            case TRIANGLE:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight << 1;
                } else {
                    return _labelWidth;
                }
            case RECTANGLE:
                return _labelHeight + 4;
            case OCTAGON:
                return getPreferredWidth();
        }
        return 0;
    }
    /* Lays out this button's contents.
     * This field's manager invokes this method during the layout
     * process to instruct this field to arrange its contents, given an
     * amount of available space.
     **/
    protected void layout(int width, int height) {
```

```
    // Update the cached font in case it has been changed.
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);

    // Calculate width.
    width = Math.min( width, getPreferredWidth() );
    // Calculate height.
    height = Math.min( height, getPreferredHeight() );
    // Set dimensions.
    setExtent( width, height );
}

/*
 * Redraws this button. The field's manager invokes this method during the
 * repainting process to instruct this field to repaint itself.
 */
protected void paint(Graphics graphics) {
    int textX, textY, textWidth;
    int w = getWidth();
    switch(_shape) {
        case TRIANGLE:
            int h = (w>>1);
            int m = (w>>1)-1;
            graphics.drawLine(0, h-1, m, 0);
            graphics.drawLine(m, 0, w-1, h-1);
            graphics.drawLine(0, h-1, w-1, h-1);

            textWidth = Math.min(_labelWidth,h);
            textX = (w - textWidth) >> 1;
            textY = h >> 1;
            break;
        case OCTAGON:
            int x = 5*w/17;
            int x2 = w-x-1;
            int x3 = w-1;
            graphics.drawLine(0, x, 0, x2);
            graphics.drawLine(x3, x, x3, x2);
            graphics.drawLine(x, 0, x2, 0);
            graphics.drawLine(x, x3, x2, x3);
            graphics.drawLine(0, x, x, 0);
            graphics.drawLine(0, x2, x, x3);
            graphics.drawLine(x2, 0, x3, x);
            graphics.drawLine(x2, x3, x3, x2);
            textWidth = Math.min(_labelWidth, w - 6);
            textX = (w-textWidth) >> 1;
            textY = (w-_labelHeight) >> 1;
            break;

        case RECTANGLE: default:
            graphics.drawRect(0, 0, w, getHeight());
            textX = 4;
            textY = 2;
            textWidth = w - 6;
            break;
    }
    graphics.drawText(_label, textX, textY, (int)( getStyle() &
        DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ),
```

```
            textWidth );
    }
}
```

Create custom context menus

Task	Steps

Create the custom context menu items.	> In your field class, create the custom context menu items.
	`private MenuItem myContextMenuItemA = new MenuItem( _resources, MENUITEM_ONE, 200000, 10) {`
	`public void run() {`
	`onMyMenuItemA();`
	`}`
	`};`
	`private MenuItem myContextMenuItemB = new MenuItem( _resources, MENUITEM_ONE, 200000, 10) {`
	`public void run() {`
	`onMyMenuItemB();`
	`}`
	`};`

Provide a context menu.	> In your main application class, override `makeContextMenu()`.
	`protected void makeContextMenu(ContextMenu contextMenu) {`
	`contextMenu.addItem(myContextMenuItemA);`
	`contextMenu.addItem(myContextMenuItemB);`
	`}`

Create the application menu.	> In your main application class, override `makeMenu()`, invoking `getLeafFieldWithFocus()` and `getContextMenu()` on the return value to determine which fields receive custom menu items.
	`protected void makeMenu(Menu menu) {`
	`Field focus = UiApplication.getUiApplication().getActiveScreen().getLeafFieldWithFocus();`
	`if (focus != null) {`
	`ContextMenu contextMenu = focus.getContextMenu();`
	`if (!contextMenu.isEmpty()) {`
	`menu.add(contextMenu);`
	`menu.addSeparator();`
	`}`
	`}`
	`}`

Code sample: Creating a custom context menu

Example: ContextMenuSample.java

```
/**
 * ContextMenuSample.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.contextmenus;

import net.rim.device.api.i18n.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;

public class ContextMenuSample extends UiApplication implements ContextMenuSampleResource
{

    private MyContextField myContextField;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
                ContextMenuSampleResource.BUNDLE_ID,
                ContextMenuSampleResource.BUNDLE_NAME);

    public static void main(String[] args) {
        ContextMenuSample app = new ContextMenuSample();
        app.enterEventDispatcher();
    }

    // Inner class to define a new field.
    private static class MyContextField extends RichTextField {
        private MenuItem myContextMenuItemA = new MenuItem(
                _resources, MENUITEM_ONE, 200000, 10) {
            public void run() {
                onMyMenuItemA();
            }
        };
        private MenuItem myContextMenuItemB = new MenuItem(
            _resources, MENUITEM_TWO, 200000, 10) {
                public void run() {
                    onMyMenuItemB();
                }
            };

        private void onMyMenuItemA() {
            // Perform an action when user selects menu item.
        }
        private void onMyMenuItemB() {
            // Perform an action when user selects menu item.
        }
```

```
        protected void makeContextMenu(ContextMenu contextMenu) {
            contextMenu.addItem(myContextMenuItemA);
            contextMenu.addItem(myContextMenuItemB);
        }

        MyContextField(String text) {
            super(text);
        }
    }

    public ContextMenuSample() {
        MainScreen mainScreen = new MainScreen();

        MyContextField myContextField = new MyContextField("Field label: ");
        mainScreen.add(myContextField);

        pushScreen(mainScreen);

    }
}
```

Create custom layout managers

Task	Steps

Create a custom layout manager.	> Extend the `Manager` class or one of its subclasses.
	`class DiagonalManager extends Manager {`
	`public DiagonalManager(long style){`
	`super(style);`
	`}`
	`...`
	`}`

Return a preferred field width.	> Override `getPreferredWidth()` so that it returns the preferred field width for the manager.
	`public int getPreferredWidth() {`
	`int width = 0;`
	`int numberOfFields = getFieldCount();`
	
	`for (int i=0; i<numberOfFields; ++i) {`
	`width += getField(i).getPreferredWidth();`
	`}`
	`return width;`
	`}`

Organize more than one TextField or Manager object.	> Override the respective `getPreferredWidth()` methods for the TextField or Manager objects.
Organize multiple TextFields horizontally.	> Override `layout()`.

Return a preferred field height.	> Override `getPreferredHeight()` so that it returns the preferred field height for the manager.
	`public int getPreferredHeight() {`
	`int height = 0;`
	`int numberOfFields = getFieldCount();`
	`for (int i=0; i<numberOfFields; ++i) {`
	`height += getField(i).getPreferredHeight();`
	`}`
	`return height;`
	`}`

Task	Steps

Specify the arrangement of the child fields.	Override `sublayout()` to retrieve the total number of fields in the manager.
	To control how each child field is added to the screen, call `setPositionChild()` and `layoutChild()` for each field that the manager contains.
	`protected void sublayout(int width, int height) {`
	`int x  = 0;`
	`int y = 0;`
	`Field field;`
	`int numberOfFields = getFieldCount();`
	`for (int i=0; i<numberOfFields; ++i) {`
	`field = getField(i);`
	`layoutChild(field, width, height);`
	`setPositionChild(field, x, y);`
	`x += field.getPreferredWidth();`
	`y += field.getPreferredHeight();`
	`}`
	`setExtent(width,height);`
	`}`

Set the size of the child fields.	> In `sublayout()`, invoke `setExtent()`.

Specify how the fields receive focus.	> Override `nextFocus()`.
	`protected int nextFocus(int direction, boolean alt) {`
	`int index = this.getFieldWithFocusIndex();`
	`if(alt) {`
	`if(direction < 0) {`
	`// action to perform if trackwheel is rolled up`
	`} else {`
	`// action to perform if trackwheel is rolled down`
	`}`
	`}`
	`if (index == this.getFieldWithFocusIndex()) {`
	`return super.nextFocus(direction, alt);`
	`} else {`
	`return index;`
	`}`
	`}`

Repaint the fields when the visible region changes.	By default, the custom manager invokes `paint()` to repaint all of the fields without regard to the clipping region. > Implement `subpaint()`.

Code sample: Creating a custom layout manager

Example: DiagonalManager.java

```
/**
 * DiagonalManager.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
```

```
package com.rim.samples.docs.custommenu;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;

class DiagonalManager extends Manager {
    public DiagonalManager(long style) {
        super(style);
    }

    public int getPreferredWidth() {
        int width = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            width += getField(i).getPreferredWidth();
        }
        return width;
    }

    public int getPreferredHeight() {
        int height = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            height += getField(i).getPreferredHeight();
        }
        return height;
    }

    protected void sublayout(int width, int height) {
        int x  = 0;
        int y = 0;
        Field field;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            field = getField(i);
            layoutChild( field, width, height );
            setPositionChild(field, x, y);
            x += field.getPreferredWidth();
            y += field.getPreferredHeight();
        }
        setExtent(width,height);
    }

   protected int nextFocus(int direction, boolean alt) {
    int index = this.getFieldWithFocusIndex();
    if(alt) {
        if(direction < 0) {
            // action to perform if trackwheel is rolled up
        } else {
            // action to perform if trackwheel is rolled down
        }
    }
    if (index == this.getFieldWithFocusIndex()) {
        return super.nextFocus(direction, alt);
   } else {
```

```
        return index;
    }
    }
}
```

Create custom lists

Task	Steps
Let users BlackBerry® device select multiple items in a list.	> Declare lists as MULTI_SELECT.

Create a callback object.	> Implement the `ListFieldCallback` interface.
	`private class ListCallback implements ListFieldCallback {`
	`// The listElements vector contain the entries in the list.`
	`private Vector listElements = new Vector();`
	`...`
	`}`

Let the field repaint a row.	> Implement `drawListRow()`, invoking `drawText()` using parameters that specify the row to paint.
	`public void drawListRow(ListField list, Graphics g, int index, int y, int w) {`
	`String text = (String)listElements.elementAt(index);`
	`g.drawText(text, 0, y, 0, w);`
	`}`

Let the field retrieve an entry from the list.	> Implement `get()`.
	`public Object get(ListField list, int index) {`
	`return listElements.elementAt(index);`
	`}`

Return a preferred width for the list.	In the implementation of `getPreferredWidth()`, return a preferred width for the list.
	`public int getPreferredWidth(ListField list) {`
	`return Graphics.getScreenWidth();`
	`}`

Assign the callback and add entries to the list.	1. Create the `ListField` and `ListCallback` objects for the list.
	`ListField myList = new ListField();`
	`ListCallback myCallback = new ListCallback();`
	2. Invoke `setCallback()` to associate the `ListFieldCallback` with the `ListField`. This association lets the callback add items to the list.
	`myList.setCallback(myCallback);`
	3. To add entries to the list, create the entries, specify an index at which to insert each entry on the `ListField` object, and then insert each `ListField` object into the `ListFieldCallback`.
	`String fieldOne = new String("Field one label");`
	`String fieldTwo = new String("Field two label");`
	`String fieldThree = new String("Field three label");`
	`myList.insert(0);`
	`myList.insert(1);`
	`myList.insert(2);`
	`myCallback.insert(fieldOne, 0);`
	`myCallback.insert(fieldTwo, 1);`
	`myCallback.insert(fieldThree, 2);`
	`mainScreen.add(myList);`

Code sample: Creating a custom list

Example: SampleListFieldCallback.java

```
/**
 * SampleListFieldCallback.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.listfields;

import java.util.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public class SampleListFieldCallback extends UiApplication {
    private ListField myList;
    public static void main(String[] args) {
        SampleListFieldCallback app = new SampleListFieldCallback();
        app.enterEventDispatcher();
    }
    private static class ListCallback implements ListFieldCallback {
        private Vector listElements = new Vector();
        public void drawListRow(
            ListField list, Graphics g, int index, int y, int w) {
            String text = (String)listElements.elementAt(index);
            g.drawText(text, 0, y, 0, w);
        }
        public Object get(ListField list, int index) {
            return listElements.elementAt(index);
        }
        public int indexOfList(ListField list, String p, int s) {
            return listElements.indexOf(p, s);
        }
        public int getPreferredWidth(ListField list) {
            return Graphics.getScreenWidth();
        }
        public void insert(String toInsert, int index) {
            listElements.addElement(toInsert);
        }
        public void erase() {
            listElements.removeAllElements();
        }
    }
    public SampleListFieldCallback() {
        MainScreen mainScreen = new MainScreen();
        myList = new ListField();
        ListCallback myCallback = new ListCallback();
        myList.setCallback(myCallback);
        String fieldOne = “ListField one”;
        String fieldTwo = “ListField two”;
        String fieldThree = “ListField three”;

        myList.insert(0);
        myCallback.insert(fieldOne, 0);
```

```
        myList.insert(1);
        myCallback.insert(fieldTwo, 1);
        myList.insert(2);
        myCallback.insert(fieldThree, 2);

        mainScreen.add(myList);

        pushScreen(mainScreen);
    }
}
```

Adding menu items to BlackBerry applications

The Application Menu Item API, in the `net.rim.blackberry.api.menuitem` package, lets you add menu items to BlackBerry® applications. The `ApplicationMenuItemRepository` class lets you add or remove menu items from BlackBerry applications.

Create a menu item

Task	Steps
Define a menu item.	> Extend the abstract `ApplicationMenuItem` class. `public class SampleMenuItem extends ApplicationMenuItem { ... }`

Specify the position of the menu item in the menu.	A higher number means that the menu item appears lower in the menu.
	> Invoke `ApplicationMenuItem()`.
	`SampleMenuItem() {`
	`super(20);`
	`}`

Specify the menu item text.	> Implement `toString()`.
	`public String toString() {`
	`return "Open the Contacts Demo application";`
	`}`

Specify the behavior of the menu item.	> Implement `run()`.
	`public Object run(Object context) {`
	`Contact c = (Contact)context; // An error occurs if this does not work.`
	`if ( c ! null ) {`
	`new ContactsDemo().enterEventDispatcher();`
	`} else {`
	`throw new IllegalStateException( "Context is null, expected a Contact instance");`
	`}`
	`Dialog.alert("Viewing a message in the messaging view");`
	`return null;`
	`}`

Register a menu item

Task	Steps
Retrieve the application menu item repository.	> Invoke `ApplicationMenuItemRepository.getInstance()`. `ApplicationMenuItemRepository repository = ApplicationMenuItemRepository.getInstance();`
Create your application menu item.	> Invoke the constructor. `TestApplicationMenuItem tami = new TestApplicationMenuItem();`
Add the menu item to the repository.	> Invoke `addMenuItem()`. `repository.addMenuItem(ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD_VIEW, tami);`

Code sample: Creating a new menu item in a BlackBerry application

The menu item appears when a BlackBerry® device user views a contact in the address book. When a BlackBerry device user clicks the menu item, the ContactsDemo application appears.

Example: DemoAppMenuItem.java

```
/**
 * DemoApplicationMenuItem.java
 * Copyright (C) 2003-2007 Research In Motion Limited.
 *
 * The following code example creates a menu item that appears when
 * a user views a contact in the address book. When a user clicks the menu item,
 * the Contacts Demo application appears.
*/

package com.rim.samples.docs.menuitem;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.component.Dialog.*;
import net.rim.blackberry.api.menuitem.*;
import net.rim.blackberry.api.pdap.*;
import javax.microedition.pim.*;



public final class DemoAppMenuItem extends Application
{
    private static final String ARG_LAUNCH_CONTACT_DEMO = “1”;
    //private static final String ARG_LAUNCH_APP2 = “2”;
    //... etc

    public static void main(String[] args) {
        if(args == null || args.length == 0)
        {
            DemoAppMenuItem app = new DemoAppMenuItem();
            app.enterEventDispatcher();
        }
        else
        {
            String appToLaunch = args[0];
            if(ARG_LAUNCH_CONTACT_DEMO.equals(appToLaunch))
            {
                new
com.rim.samples.docs.contactsdemo.ContactsDemo().enterEventDispatcher();

            }

            //add more else ifs here
        }
    }
```

```
    DemoAppMenuItem() {
        long locationToAddMenuItem =
ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD_VIEW;
        addMenuItem(ARG_LAUNCH_CONTACT_DEMO, locationToAddMenuItem, new
ContactsDemoMenuItem());

        System.exit(0);
    }

    private static void addMenuItem(String argOfAppl, long location, ApplicationMenuItem
applMenuItem)
    {
        ApplicationMenuItemRepository amir = ApplicationMenuItemRepository.getInstance();
        ApplicationDescriptor app = ApplicationDescriptor.currentApplicationDescriptor();

        //set the argument so that we know which app we want to have launched
        app = new ApplicationDescriptor(app, new String[]{ARG_LAUNCH_CONTACT_DEMO});

        amir.addMenuItem(location, applMenuItem, app);
    }

   /**
    * Create the menu item classes here
    */
    private static class ContactsDemoMenuItem extends ApplicationMenuItem {
        ContactsDemoMenuItem() {
            super(20);
        }

        public String toString() {
            return “Open the Contacts Demo”;
        }

        public Object run(Object context) {
            BlackBerryContact c = (BlackBerryContact)context; //an error if this doesn’t
work
            if ( c != null ) {
                Application.getApplication().requestForeground();
                //on invokation, will call the main method of this app. with argument as
specified in addMenuItem
            } else {
                throw new IllegalStateException( “Context is null, expected a Contact
instance”);
            }
            return null;
        }
    }
}
```

Arrange UI components

To arrange components on a screen, use BlackBerry® API layout managers.

The following four classes extend the `Manager` class to provide predefined layout managers:

- `VerticalFieldManager`
- `HorizontalFieldManager`
- `FlowFieldManager`
- `DialogFieldManager`

To create a custom layout manager, extend `Manager`.

Define a layout manager

Task	Steps

Create a layout manager.	On an instance of a `Screen`, complete the following actions:
	1. Instantiate the appropriate `Manager` subclass.
	`VerticalFieldManager vfm = new VerticalFieldManager(Manager.VERTICAL_SCROLL);`
	2. Add UI components to the layout manager.
	`vfm.add(bitmapField);`
	`vfm.add(bitmapField2);`
	3. Add the layout manager to the screen.
	`mainScreen.add(vfm)`

Set field focus and navigation

UI `EventListeners` let applications respond to a change to a UI object.

Listen for field focus changes

1. Implement `FocusChangeListener`. Your implementation of `FocusChangeListener` should specify what action occurs when the field gains, loses, or changes the focus by implementing `focusChanged()`.
2. Assign your implementation to a `Field` by invoking `setChangeListener()`.

```
private class FocusListener implements FocusChangeListener {
public void focusChanged(Field field, int eventType) {
if (eventType == FOCUS_GAINED) {
// Perform action when this field gains the focus.}
if (eventType == FOCUS_CHANGED) {
// Perform action when the focus changes for this field.}
if (eventType == FOCUS_LOST) {
// Perform action when this field loses focus.}
}
}
FocusListener myFocusChangeListener = new FocusListener();
myField.setFocusListener(myFocusChangeListener);
```

Respond to UI events

Task	Steps
Respond to UI navigation events.	> Manage navigation events by extending the `net.rim.device.api.ui.Screen class` (or one of its subclasses) and overriding the following navigation methods: • `navigationClick(int status, int time)` • `navigationUnclick(int status, int time)` • `navigationMovement(int dx, int dy, int status, int time)` • When you create a new UI application, use the new `Screen` navigation methods and avoid using the `TrackwheelListener`. • If the existing UI application implements the `TrackwheelListener`, update the application to use the new `Screen` navigation methods.

Task	Steps

Interpret the status parameter of the navigation methods.	> In your implementation of one of the `navigationClick`, `navigationUnclick`, or `navigationMovement` methods of the `Screen` or `Field` classes, perform a bitwise AND operation on the **status** parameter to yield more information about the event. For example, to determine the type of input mechanism that triggered an event, in your implementation of the `navigationClick(int status, int time)` method, create code such as the following:
	`public boolean navigationClick(int status, int time) {`
	`if ((status & KeypadListener.STATUS_TRACKWHEEL) == KeypadListener.STATUS_TRACKWHEEL) {`
	`//Input came from the trackwheel`
	`} else if ((status & KeypadListener.STATUS_FOUR_WAY) == KeypadListener.STATUS_FOUR_WAY) {`
	`//Input came from a four way navigation input device`
	`}`
	`return super.navigationClick(status, time); }`
	See the *API Reference* for the class `net.rim.device.api.system.KeypadListener` for a listing of other status modifiers.

Respond to BlackBerry® device user interaction.	> Use the `Screen` class and its subclasses to provide a menu for the BlackBerry device users to perform actions.
Provide screen navigation when using a `FullScreen` or `Screen`.	Creating a `MainScreen` object provides default navigation to your application. Avoid using buttons or other UI elements that take up space on the screen. > Specify the DEFAULT_MENU and DEFAULT_CLOSE parameters in the constructor to provide default navigation. `FullScreen fullScreen = new FullScreen(DEFAULT_MENU | DEFAULT_CLOSE);`
Provice menu support.	> Extend the `Screen` class.
Provide menu support in an application that uses the `TrackwheelClick()` method of the `TrackwheelListener`.	1. Update your application to use an extension of the `Screen` class. 2. In the constructor of your `Screen` class extension, make sure to invoke the `Screen` class constructor using the DEFAULT_MENU property. 3. Make sure your extension of the `makeMenu()` method of the `Screen` class invokes `Screen.makeMenu()` and adds the required menu items for the current UI application.

Define a layout manager

Task	Steps

Create a layout manager.	On an instance of a `Screen`, complete the following actions:
	1. Instantiate the appropriate `Manager` subclass.
	`VerticalFieldManager vfm = new VerticalFieldManager(Manager.VERTICAL_SCROLL);`
	2. Add UI components to the layout manager.
	`vfm.add(bitmapField);`
	`vfm.add(bitmapField2);`
	3. Add the layout manager to the screen.
	`mainScreen.add(vfm)`

Set field focus and navigation

UI `EventListeners` let applications respond to a change to a UI object.

Listen for field focus changes

1. Implement `FocusChangeListener`. Your implementation of `FocusChangeListener` should specify what action occurs when the field gains, loses, or changes the focus by implementing `focusChanged()`.
2. Assign your implementation to a `Field` by invoking `setChangeListener()`.

```
private class FocusListener implements FocusChangeListener {
public void focusChanged(Field field, int eventType) {
if (eventType == FOCUS_GAINED) {
// Perform action when this field gains the focus.}
if (eventType == FOCUS_CHANGED) {
// Perform action when the focus changes for this field.}
if (eventType == FOCUS_LOST) {
// Perform action when this field loses focus.}
}
}
FocusListener myFocusChangeListener = new FocusListener();
myField.setFocusListener(myFocusChangeListener);
```

Respond to UI events

Task	Steps
Respond to UI navigation events.	> Manage navigation events by extending the `net.rim.device.api.ui.Screen class` (or one of its subclasses) and overriding the following navigation methods: • `navigationClick(int status, int time)` • `navigationUnclick(int status, int time)` • `navigationMovement(int dx, int dy, int status, int time)` • When you create a new UI application, use the new `Screen` navigation methods and avoid using the `TrackwheelListener`. • If the existing UI application implements the `TrackwheelListener`, update the application to use the new `Screen` navigation methods.

Manage foreground events

The system calls `Application.activate()` when it brings an application to the foreground.

Manage drawing areas

The `Graphics` object represents the entire drawing surface that is available to the application. To limit this area, divide it into `XYRect` objects. Each `XYPoint` represents a point on the screen, which is composed of an X co-ordinate and a Y co-ordinate.

Task	Steps

Create rectangular clipping areas.	1. Create an instance of an `XYPoint` object and an `XYRect` object.
	`XYPoint bottomRight = new XYPoint(50, 50);`
	`XYRect rectangle = new XYRect(topLeft, bottomRight);`
	`XYPoint topLeft = new XYPoint(10, 10);`
	2. Invoke `pushContext()` or `pushRegion()`.
	3. When you make drawing calls with `pushContext()`, specify that the region origin should not adjust the drawing offset.
	`graphics.pushContext(rectangle, 0, 0);`
	`graphics.fillRect(10, 10, 30, 30);`
	`graphics.drawRect(15, 15, 30, 30);`
	`graphics.popContext();`
	4. When you invoke drawing methods by first calling `pushRegion()`, specify that the region origin should adjust the drawing offset.
	`graphics.pushRegion(rectangle);`
	`graphics.fillRect(10, 10, 30, 30);`
	`graphics.drawRect(15, 15, 30, 30);`
	`graphics.popRegion();`

Invert an area.	1. Invert a specified `XYRect` object.
	2. Specify the portion of the `Graphics` object to push onto the stack.
	3. After you invoke `pushContext()` (or `pushRegion()`), provide the portion of the `Graphics` object to invert.
	`graphics.pushContext(rectangle);`
	`graphics.invert(rectangle); // invert the entire XYRect object`
	`graphics.popContext();`

Translate an area.	> Invoke `translate()`. The `XYRect` translates from its origin of (1, 1) to an origin of (20, 20). After translation, the bottom portion of the `XYRect` object extends past the bounds of the graphics context and clips it.
	`XYRect rectangle = new XYRect(1, 1, 100, 100);`
	`XYPoint newLocation = new XYPoint(20, 20);`
	`rectangle.translate(newLocation);`

2

Using graphics and multimedia

Using images
Drawing and rendering images
Using audio
Using rich media

Using images

Use raw images

Task	Steps
Allow applications to use raw image data.	> To retrieve raw image data from a specified region of a bitmap and store the data in an integer array, invoke `Bitmap.getARGB()`. `void getARGB(int[] argbData, int offset, int scanLength, int x, int y, int width, int height);`

Retrieve image data.	1. Initialize an integer array.
	`Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);`
	`int[] argb = new int[original.getWidth() * original.getHeight()];`
	2. To store the raw image data of the new or predefined bitmap in the integer array, invoke `Bitmap.getARGB()`.
	`original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(), original.getHeight());`

Compare two images to see if they are identical.	> Invoke `Bitmap.equals()`.
	`if(restored.equals(original)) {`
	`System.out.println("Success! Bitmap renders correctly with RGB data.");`
	`} else if(!restored.equals(original)) {`
	`System.out.println("Bitmap rendered incorrectly with RGB data.");`
	`}`

Use encoded images

Task	Steps

Access an image.	1. Save an image to the project folder or subfolder.
	2. Add the image to the project in the BlackBerry® Integrated Development Environment.
	3. Invoke `Class.getResourceAsStream()` to retrieve the image as an input stream of bytes.
	`private InputStream input;`
	`...`
	`try {`
	`input = Class.forName("com.rim.samples.docs.imagedemo.ImageDemo").`
	`getResourceAsStream("/images/example.png");`
	`} catch (ClassNotFoundException e) {`
	`System.out.println("Class not found");`
	`}`

Encode an image.	1. Invoke `EncodedImage.createEncodedImage()`. This method creates an instance of `EncodedImage` using the raw image data in the byte array.
	2. Check for an `IllegalArgumentException`, which `EncodedImage.createEncodedImage()` throws if the byte array that you provide as a parameter does not contain a recognized image format.
	`private byte[] data = new byte[2430]; // Store the contents of the image file.`
	`try {`
	`input.read(data); // Read the image data into the byte array.`
	`} catch (IOException e) {`
	`// Handle exception.`
	`}`
	`try {`
	`EncodedImage image = EncodedImage.createEncodedImage(data, 0, data.length);`
	`} catch (IllegalArgumentException iae) {`
	`System.out.println("Image format not recognized.");`
	`}`

Display an encoded image.	1. To assign the encoded image to a `BitmapField`, invoke `BitmapField.setImage()`.
	`BitmapField field = new BitmapField();`
	`field.setImage(image);`
	2. To add the `BitmapField` to the screen, invoke `add()`.
	`add(field);`

Set the decoding mode.	1. Invoke `EncodedImage.setDecodeMode()`. 2. Provide one of the following modes as a parameter to the method: • `DECODE_ALPHA`: decodes an alpha channel, if one exists (this is the default mode) • `DECODE_NATIVE`: forces the application to decode the bitmap to the native bitmap type of the BlackBerry device software application • `DECODE_READONLY`: marks the decoded bitmap as read-only
Set the image display size.	> Invoke `EncodedImage.setScale()`. `The inverse of the integer that the scale parameter specifies scales the image. For example, if you set the scaling factor to 2, the image decodes at 50% of its original size.`

Code sample: Using a raw image to recreate an encoded image

Example: ImageDemo.java

```
/**
 * ImageDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.imagedemo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.io.*;

/* The ImageDemo.java sample retrieves raw data from an image that
   is included in its project, and then uses that raw data to
   recreate an EncodedImage. */
public class ImageDemo extends UiApplication {
    public static void main(String[] args) {
        ImageDemo app = new ImageDemo();
        app.enterEventDispatcher();
    }
    public ImageDemo() {
        pushScreen(new ImageDemoScreen());
    }
}

final class ImageDemoScreen extends MainScreen {
    private static final int IMAGE_SIZE = 2430;
    private InputStream input;
    private byte[] data = new byte[IMAGE_SIZE];

    public ImageDemoScreen() {
        super();
        setTitle(new LabelField("Image Demo Sample"));

        try {
            input =
Class.forName("com.rim.samples.docs.imagedemo.ImageDemo").getResourceAsStream("/images/
hellokitty.png");
        } catch (ClassNotFoundException e) {
            System.out.println("Class not found");
        }

        if(input == null) {
            System.out.println("Error: input stream is not initialized.");
        } else if (input != null) {
            System.out.println("OK: input stream is initialized.");
            try {
                int code = input.read(data);
                System.out.println("Total number of bytes read into buffer: " + code + "
.");
            } catch (IOException e) {
                // Handle exception.
            }
```

```
            try {
                EncodedImage image = EncodedImage.createEncodedImage(data, 0, data.length);
                 add(new BitmapField(image.getBitmap()));
            } catch (IllegalArgumentException iae) {
                System.out.println("Image format not recognized.");
            }
        }
    }
}
```

Drawing and rendering images

Position an image

Task	Steps

Use an individual field.	1. Invoke the `Graphics()` constructor.
	`Bitmap surface = new Bitmap(100, 100);`
	`BitmapField surfaceField = new BitmapField(surface);`
	`Graphics graphics = new Graphics(surface);`
	2. Add the `BitmapField` to the screen.
	`mainScreen.add(surfaceField);`

Use the whole screen.	1. Invoke `Screen.getGraphics()`.
	`Graphics graphics = Screen.getGraphics();`
	2. Make sure your methods perform their drawing functions within the boundaries of the screen.
	`graphics.fillRect(10, 10, 30, 30);`
	`graphics.drawRect(15, 15, 30, 30);`

Draw an image in color

Task	Steps
Determine whether the BlackBerry® device supports color display.	> Invoke `Graphics.isColor()`.
Determine the number of colors that the BlackBerry device supports.	> Invoke `Graphics.getNumColors()`.
Set the pixel transparency in the drawing area.	1. Invoke one of the following methods: • `Graphics.setGlobalApha()` • `Graphics.getGlobalAlpha()` 2. Define a global alpha value within the following range: • 0 (0x0000): completely transparent • 255 (0x00FF): fully opaque

Task	Steps
Determine the raster operations that the application supports.	1. Invoke `Graphics.isRopSupported(int)`. 2. Provide one of the following constants as a parameter: • `ROP_CONST_GLOBALALPHA`: blends the constant foreground color using a constant global alpha value with destination pixels • `ROP_SRC_GLOBALALPHA`: blends a source bitmap using a constant global alpha value with destination pixels

Draw a set of shaded, filled paths.	> Invoke `Graphics.drawShadedFilledPath()`:
	`public void drawShadedFilledPath(int[] xPts, int[] yPts, byte[] pointTypes, int[] colors, int[] offsets);`
	The following example draws a path that blends from blue to red:
	`Bitmap surface = new Bitmap(240, 160);`
	`BitmapField surfaceField = new BitmapField(surface);`
	`add(surfaceField);`
	`Graphics graphics = new Graphics(surface);`
	`int[] X_PTS = { 0, 0, 240, 240 };`
	`int[] Y_PTS = { 20, 50, 50, 20 };`
	`int[] drawColors = { 0x0000CC, 0x0000CC, 0xCC0000, 0xCC0000 };`
	`try {`
	`graphics.drawShadedFilledPath(X_PTS, Y_PTS, null, drawColors, null);`
	`} catch (IllegalArgumentException iae) {`
	`System.out.println("Bad arguments.");`
	`}`

Turn a drawing style on.	> Invoke `Graphics.setDrawingStyle(int drawStyle, boolean on)`.
Turn a drawing style off.	> Invoke `Graphics.setDrawingStyle(int drawStyle, boolean off)`.
Determine if a drawing style is set.	> Invoke `Graphics.isDrawingStyleSet(int drawStyle)`.
Use a monochrome bitmap as a stamp.	> To let applications use monochrome bitmaps as stamps by rendering the nontransparent region in color, use the `STAMP_MONOCHROME` option. This option applies to bitmaps that are 1 bit and have alpha defined. `BitmapField field = new BitmapField(original, BitmapField.STAMP_MONOCHROME);`

Draw an image on an empty bitmap.	1. Create an empty bitmap. The example that follows copies the type and size from an existing bitmap.
	`Bitmap restored = new Bitmap(original.getType(), original.getWidth(), original.getHeight());`
	2. Create a `Graphics` object using the empty bitmap as the drawing surface.
	`Graphics graphics = new Graphics(restored);`
	3. To draw a new image using raw data retrieved from the original, invoke `Graphics.drawRGB()`.
	`try {`
	`graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0, restored.getWidth(), restored.getHeight());`
	`} catch(Exception e) {`
	`System.out.println("Error occurred during drawing: " + e);`
	`}`

Code sample: Drawing a new bitmap using an existing bitmap

Example: DrawDemo.java

```
/*
 * DrawDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.drawing;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

/* The DrawDemo.java sample retrieves raw data from a predefined bitmap
   image, and then draws a new bitmap using the data. It then displays
   the original and restored images. */
public class DrawDemo extends UiApplication {

    public static void main(String[] args) {
        DrawDemo app = new DrawDemo();
        app.enterEventDispatcher();
    }

    public DrawDemo() {
        pushScreen(new DrawDemoScreen());
    }
}

final class DrawDemoScreen extends MainScreen {
    public DrawDemoScreen() {
        super();

        LabelField title = new LabelField("UI Demo", LabelField.USE_ALL_WIDTH);
        setTitle(title);

        Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
        Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
                        original.getHeight());

        Graphics graphics = new Graphics(restored);

        // Retrieve raw data from original image.
        int[] argb = new int[original.getWidth() * original.getHeight()];
        original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(),
                        original.getHeight());

        // Draw new image using raw data retrieved from original image.
        try {
            graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0, restored.getWidth(),
                restored.getHeight());
        } catch(Exception e) {
```

```
            System.out.println("Error occurred during drawing: " + e);
        }


        if(restored.equals(original)) {
            System.out.println("Success! Bitmap renders correctly with RGB data.");
        } else if(!restored.equals(original)) {
            System.out.println("Bitmap rendered incorrectly with RGB data.");
        }

        BitmapField field1 = new BitmapField(original, BitmapField.STAMP_MONOCHROME);
        BitmapField field2 = new BitmapField(restored);
        add(new LabelField("Original bitmap: "));
        add(field1);
        add(new LabelField("Restored bitmap: "));
        add(field2);
    }
}
```

Using audio

Start the media player from the BlackBerry Browser

1. Invoke `Browser.getDefaultSession()`.

   ```
   BrowserSession soundclip = Browser.getDefaultSession();
   ```

2. Invoke `BrowserSession.displaypage()`.

   ```
   soundclip.displayPage( "file:///SDCard/BlackBerry/music/TarzanYell.mp3" );
   ```

Create a media player

To play audio on a BlackBerry® device, use the Mobile Media API to create a media player and then add functionality to it.

Task	Steps
Create a player for a sequence of tones.	> Use the ToneControl to permit playback of a BlackBerry device user-defined sequence of tones in an unvarying pitch. Tempo is the bpm with 1 beat equal to 1/4 note. You determine the tempo by multiplying the tempo modifier by 4 to keep it within the byte range of 1 to 127. Tempos in the range of 20 bpm to 508 bpm equate to a tempo modifier range of 5 to 127.
Create a player for media from a URL.	> Invoke `Manager.createPlayer(String locator)`. The string must use URI syntax that describes the media content.

Task	Steps

Create a player for media from an input stream.	1. Invoke `Manager.createPlayer(InputStream stream, String type).`
	The type parameter represents the input media content type.
	2. Check for a `MediaException` if null is the content type.
	`RecordStore recSt;`
	`int recId;`
	`try {`
	`InputStream inpStr = new`
	`ByteArrayInputStream((store.getRecord(recId));`
	`Player p = Manager.createPlayer(inpStr, "audio/mpeg");`
	`p.start();`
	`} catch (IOException ioEx) {`
	`} catch (MediaException meEx) {}`

Code sample: Creating a player for a sequence of tones

Example: Sequence of tones

```
// "Mary Had A Little Lamb" has "ABAC" structure
// Use block to repeat "A" section

byte tempo = 30; // 30 x 4 = tempo of 120 bpm
byte duration =  8; // Note length 8 (quaver) = 1/8th of a note duration

byte C4 = ToneControl.C4; // C note value = 60 (middle C)
byte D4 = (byte)(C4 + 2); // D note value = 62 (a whole step)
byte E4 = (byte)(C4 + 4); // E note value = 64 (a major third)
byte G4 = (byte)(C4 + 7); // G note value = 67 (a fifth)
byte rest = ToneControl.SILENCE; // rest

byte[] mySequence = {
   ToneControl.VERSION, 1,     // version 1
   ToneControl.TEMPO, tempo,   // set tempo
   //
   // Start define "A" section
   ToneControl.BLOCK_START, 0,
   //
   // Content of "A" section
   E4, duration, D4, duration, C4, duration, E4, duration,
   E4, duration, E4, duration, E4, duration, rest, duration,
   //
   // End define "A" section
   ToneControl.BLOCK_END, 0,
   //
   // Play "A" section
   ToneControl.PLAY_BLOCK, 0,
   //
   // Play "B" section
   D4, duration, D4, duration, D4, duration, rest, duration,
   E4, duration, G4, duration, G4, duration, rest, duration,
   //
   // Repeat "A" section
   ToneControl.PLAY_BLOCK, 0,
   //
```

```
    // Play "C" section
    D4, duration, D4, duration, E4, duration, D4, duration, C4, duration
};

try{
    Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
    p.realize();
    ToneControl c = (ToneControl)p.getControl("ToneControl");
    c.setSequence(mySequence);
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) { }
```

Code sample: Creating a player for media from an input stream

Example: Play an MP3 audio file

```
//First we determine the supported content types
String types[] = Manager.getSupportedContentTypes(null);
for (int cnt = types.length - 1; cnt >= 0; --cnt) {

    if (types[cnt].equals("audio/mpeg")) {

        try {
        //Retrieve the MP3 file
        Class clazz = Class.forName("com.rim.samples.AudioDemo");
        InputStream is = clazz.getResourceAsStream("/ TarzanYell.mp3");

        //Create an instance of the player from the InputStream
        Player player = javax.microedition.media.Manager.createPlayer
        (is, "audio/mpeg");

        player.realize();
        player.prefetch();

        //start the player
        player.start();
        } catch (Exception ex) { }
    }

        else if (types[cnt].equals("audio/x-wav ")) {
        //this is where you would play wav files
        }

        else if (types[cnt].equals("audio/midi ")) {
        //this is where you would play midi files
        }

    }
}
```

Play media

Task	Steps
Prepare the media player.	1. Invoke `Player.realize()`. 2. Invoke `Player.prefetch()`.
Start the media player.	> Invoke `Player.start()`. The `Player` returns to the Prefetched state when you invoke `Player.stop()` or when it reaches the end of the media file. `try { Player p = Manager.createPlayer("http://www.test.rim.net/abc.wav"); p.start(); } catch (MediaException pe) { } catch (IOException ioe) { }`
Determine the controls that a media player supports.	1. Invoke `Player.getControls()`. 2. To provide additional functionality for a media player, use one or more of the controls that the media player supports. You can use the same object to access multiple controls: for example, one object can be both a VolumeControl and a ToneControl. The `javax.microedition.media` package contains a number of Control interfaces. See the *API Reference* in the BlackBerry® Java Development Environment for more information about the `javax.microedition.media` package.
Adjust the volume of the media player.	1. Invoke `VolumeControl()`. 2. Define a volume value in the following range: • 0: no volume • 100: maximum volume level The `PlayerListener` sends a `VOLUME_CHANGED` event when its state changes.
Close the media player.	> Invoke `Player.stop()`.

Listen for media player events

Task	Steps
Listen for changes to the media player state.	1. Implement `PlayerListener`. 2. To register the player listener, invoke `addPlayerListener`. `private void doPlay() throws IOException, MediaException {Player p = Manager.createPlayer("http://www.rim.com/rim.mp3"); p.addPlayerListener(this); p.realize(); p.prefetch(); p.start(); }`

Task	Steps

Send a media player event to a registered player listener.	> Invoke `playerUpdate(Player player, String event, Object eventData)`.
	`public void playerUpdate(Player player,String event, Object eventData)`
	`{// Release resources`
	`player.close();`
	
	`if ( event == PlayerListener.END_OF_MEDIA )`
	`// Add code for actions if the end of media is reached.`
	`}`

Using rich media

Playing rich media content

To play rich media content, use the following classes:

- To retrieve .pme content from BlackBerry® devices or networks, use methods from the `MediaManager` class.
- To play .pme content that exists on BlackBerry devices, use methods from the `MediaPlayer` class.

Download rich media content

1. Create a `MediaManager` object.
2. Invoke `MediaManager.createMedia()`.

   The first time that you invoke `MediaManager.createMedia()`, the URL must be absolute, unless you first invoke `MediaManager.setProperty("URI_BASE", <base_url>)` to set a base URL. When you invoke `createMedia()` subsequently, the URL that the method used previously is the base.

```
MediaManager manager = new MediaManager();
try {
Object media = manager.createMedia("http://webserver/sample.pme");
} catch (IOException ioe) {
System.out.println("Error: requested content was not downloaded.");
} catch (MediaException me) {
System.out.println("Error: " + me.getCode()); }
```

Play rich media content

Task	Steps

Set the .pme object for playback.	> Invoke `MediaPlayer.setMedia()`.
	`MediaPlayer player = new MediaPlayer();`
	`try {`
	`player.setMedia(media);`
	`} catch (MediaException me) {`
	`System.out.println("Error: requested content type is not supported.");`
	`}`

Task	Steps
Retrieve a UI object that displays rich media content.	1. Invoke `MediaPlayer.getUI()`. 2. Cast the object that `getUI()` returns as a `Field`, and add it to a `Screen` for display. `add((Field)player.getUI());`

Play rich media content.	1. Check the media player state.
	2. Invoke `MediaPlayer.start()`.
	`if(player.getState() == MediaPlayer.REALIZED) {`
	`try {`
	`player.start();`
	`} catch(MediaException me) {`
	`System.out.println("Error occurred during media playback: " +`
	`me.getCode() + me.getMessage());`
	`}`
	`}`

Code sample: Retrieving and displaying a rich media file

The MediaSample.java code sample retrieves a .pme file from a web server and displays it on the BlackBerry® device.

Example: MediaSample.java

```
/**
 * MediaSample.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;

public class MediaSample extends UiApplication {

    public static void main(String[] args) {
        MediaSample app = new MediaSample();
        app.enterEventDispatcher();
    }
    public MediaSample() {
        pushScreen(new MediaSampleScreen());
    }
    final static class MediaSampleScreen extends MainScreen {
        public MediaSampleScreen() {
            super();
            LabelField title = new LabelField(“Media Sample”, LabelField.ELLIPSIS
            | LabelField.USE_ALL_WIDTH);
            setTitle(title);
```

```
            MediaPlayer player = new MediaPlayer();
            MediaManager manager = new MediaManager();

            try {
                Object media = manager.createMedia("http://webserver/SVGFILE.pme");
                player.setMedia(media);
            } catch (IOException ioe) {
            } catch (MediaException me) {
                System.out.println("Error during media loading: ");
                System.out.println(me.getCode());
                System.out.println(me.getMessage());
            }
            add((Field)player.getUI());
            try {
                player.start();
            } catch(MediaException me) {
                System.out.println("Error occured during media playback: ");
                System.out.println(me.getCode());
                System.out.println(me.getMessage());
            }
        }
    }
}
```

Listen for rich media events

Task	Steps

Listen for media engine events.	1. Implement the `MediaListener` interface to let your application listen for media engine events.
	`public final class MediaListenerImpl implements MediaListener {`
	2. Implement `mediaEvent()` to manage all possible media events.
	`public void mediaEvent(Object sender, int event, int eventParam, Object data)`
	`{`
	`switch(event) {`
	`case MEDIA_REQUESTED:`
	`// Perform action.`
	`break;`
	`case MEDIA_COMPLETE:`
	`// Perform action.`
	`break;`
	`case MEDIA_REALIZED:`
	`// Perform action.`
	`break;`
	`case MEDIA_IO:`
	`// Perform action.`
	`break;`
	`}`
	`}`
	`}`

Task	Steps

Register the listener.	> Invoke `addMediaListener()` on the `MediaPlayer` and `MediaManager` objects.
	`private MediaListenerImpl _listener = new MediaListenerImpl();`
	`private MediaPlayer player = new MediaPlayer();`
	`private MediaManager manager = new MediaManager();`
	`player.addMediaListener(_listener);`
	`manager.addMediaListener(_listener);`

Load the content in the background, and play it when the download is complete.	1. To download the content for future playback, invoke `MediaManager.createMediaLater()`.
	2. In `MediaListener.mediaEvent()`, add code to manage the `MEDIA_REALIZED` event that occurs when the content that the application downloads finishes loading on the BlackBerry® device.
	3. To register the content that the data parameter specifies, invoke `MediaPlayer.setMedia(data)`.
	4. To start the playback, invoke `MediaPlayer.start()`.
	`manager.createMediaLater("http://webserver/sample.pme");`
	`public void mediaEvent(Object sender, int event, int eventParam, Object data) {`
	`switch(event) {`
	`...`
	`case MEDIA_REALIZED:`
	`try {`
	`player.setMedia(data);`
	`player.start();`
	`} catch(MediaException me) {`
	`System.out.println("Error playing media" + me.getCode() +`
	`me.getMessage());`
	`}`
	`break;`
	`}`
	`}`

Track the progress of a download.	1. Extend the `net.rim.plazmic.mediaengine.io.LoadingStatus` class. 2. In your implementation of `mediaEvent()`, when the `MEDIA_IO` event occurs, cast the `Object` in the `data` parameter to a `LoadingStatus` object. 3. To retrieve the download status and manage each status, invoke `LoadingStatus.getStatus()`. 4. For each normal status, print a message to the console.
Manage a failed download.	For the LOADING_FAILED status, perform the following actions: 1. To retrieve the error code, invoke `LoadingStatus.getCode()`. 2. To retrieve the detailed message, invoke `LoadingStatus.getMessage()`. 3. To retrieve the URL string of the content, invoke `LoadingStatus.getSource()`.

Code sample: Managing rich media content download events

Example: Managing rich media content download events

```
public void mediaEvent(Object sender, int event, int eventParam, Object data) {
    switch(event) {
        ...
        case MEDIA_IO: {
            LoadingStatus s = (LoadingStatus)data;
        }
```

```
        ...
        break;
    }
    break;
    ...
    switch(s.getStatus()) {
        case LoadingStatus.LOADING_STARTED:
            System.out.println("Loading in progress");
        break;
        case LoadingStatus.LOADING_READING:
            System.out.println("Parsing in progress");
        break;
        case LoadingStatus.LOADING_FINISHED:
            System.out.println("Loading completed");
        break;
        case LoadingStatus.LOADING_FAILED:
            String errorName = null;
            int code = s.getCode();
            switch (code) {
                case MediaException.INVALID_HEADER:
                errorName =  "Invalid header" + "\n" +  s.getSource();
                break;
                case MediaException.REQUEST_TIMED_OUT:
                errorName = "Request timed out" + "\n" +
                s.getSource();
                break;
                case MediaException.INTERRUPTED_DOWNLOAD:
                break;
                case MediaException.UNSUPPORTED_TYPE:
                errorName = "Unsupported type" + s.getMessage() + "\n" + s.getSource();
                break;
                default: {
                if (code > 200) {
                // A code > 200 indicates an HTTP error
                errorName = "URL not found";
                } else {
                // default unidentified error
                errorName = "Loading Failed";
                }
                errorName += "\n" + s.getSource() + "\n" + s.getCode()
                + ": " + s.getMessage();
                break;
            }
        }
        System.out.println(errorName);
        break;
    } // End switch s.getStatus().
    break;
}
```

Code sample: Implementing a listener to download rich media content

The MediaSample2.java code sample implements a listener to download media content in the background and display the download status to the console.

Example: MediaSample2.java

```
/**
 * MediaSample2.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class MediaSample2 extends UiApplication {
    private MediaPlayer player = new MediaPlayer();
    private MediaManager manager = new MediaManager();
    private MediaListenerImpl _listener = new MediaListenerImpl();

    private MediaSample2Screen _screen;

    public static void main(String[] args) {
         MediaSample2 app = new MediaSample2();
         app.enterEventDispatcher();
    }

    public MediaSample2() {
        _screen = new MediaSample2Screen();
        pushScreen(_screen);
    }

    public final class MediaListenerImpl implements MediaListener {
        public void mediaEvent(Object sender, int event, int eventParam, Object data) {
            switch(event) {
                case MEDIA_REQUESTED:
                    System.out.println(“Media requested”);
                    break;
                case MEDIA_COMPLETE:
                    System.out.println(“Media completed”);
                    break;
                case MEDIA_REALIZED:
                    try {
                        player.setMedia(data);
                        player.start();
                    }
                    catch(MediaException me) {
                       System.out.println(“Error during media loading: “ + me.getCode() +
me.getMessage());
                    }
                    break;
                case MEDIA_IO: {
                    LoadingStatus s = (LoadingStatus)data;
                    switch(s.getStatus()) {
```

```
                        case LoadingStatus.LOADING_STARTED:
                            System.out.println("Loading in progress");
                            break;
                        case LoadingStatus.LOADING_READING:
                            System.out.println("Parsing in progress");
                            break;
                        case LoadingStatus.LOADING_FINISHED:
                            System.out.println("Loading completed");
                            break;
                        case LoadingStatus.LOADING_FAILED:
                            String errorName = null;
                            int code = s.getCode();
                            switch (code) {
                                case MediaException.INVALID_HEADER:
                                    errorName =  "Invalid header" + "\n" +  s.getSource();
                                    break;

                                case MediaException.REQUEST_TIMED_OUT:
                                   errorName = "Request timed out" + "\n" +  s.getSource();
                                    break;

                                case MediaException.INTERRUPTED_DOWNLOAD:
                                    break;

                                case MediaException.UNSUPPORTED_TYPE:
                                   errorName = "Unsupported type" + s.getMessage() + "\n"
+ s.getSource();
                                    break;

                                default: {
                                    if (code > 200) {
                                        // A code > 200 indicates an HTTP error.
                                        errorName = "URL not found";
                                    }
                                    else {
                                        // Default unidentified error.
                                        errorName = "Loading Failed";
                                    }
                                    errorName += "\n" + s.getSource() + "\n"
                                        + s.getCode() + ": " + s.getMessage();
                                    break;
                                }

                            }
                            System.out.println(errorName);
                            break;
                    } // End switch s.getStatus().
                    break;
                }
            }
        }
    }
    final class MediaSample2Screen extends MainScreen {
        public MediaSample2Screen() {
            super();
            LabelField title = new LabelField("Media Sample", LabelField.ELLIPSIS
                | LabelField.USE_ALL_WIDTH);
            setTitle(title);
```

```
            player.addMediaListener(_listener);
            manager.addMediaListener(_listener);

            // Change this to the location of a test .pme file.
            manager.createMediaLater("http://test.rim.com/SVGBS0001.pme");
            add((Field)player.getUI());
        }
    }
}
```

Create a custom connector for rich media connections

To add support for a custom protocol or to override default behavior, create a custom `Connector`.

Task	Steps
Implement a custom connector.	> Implement the `net.rim.plazmic.mediaengine.io.Connector` interface.
Return an input stream to read content from a URI.	> Implement `InputStream getInputStream(String, ConnectionInfo)`.
Set custom connector properties.	> Implement `setProperty(String, String)`.
Release the custom connection.	> Implement `void releaseConnection(ConnectionInfo)`.

Register a custom connector.	> In your main method, invoke `MediaManager.setConnector()`.
	`MediaManager manager = new MediaManager();`
	`manager.setConnector(new CustomPMEConnector(manager.getDefaultConnector()));`

Code sample: Implementing a custom connector

The CustomPMEConnector.java code sample provides a framework for implementing a custom connector.

Example: CustomPMEConnector.java

```
/*
 * CustomPMEConnector.java
 * Copyright (C) 2003-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class CustomPMEConnector implements Connector {

    private Connector delegate;
    private InputStream input;

    CustomPMEConnector(Connector delegate) {
        this.delegate = delegate;
```

```
    }
    public InputStream getInputStream(String uri, ConnectionInfo info)
        throws IOException, MediaException {
        if (uri.startsWith("myprotocol://")) {
            // Perform special tasks.
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");
            // OpenMyInputStream() is a custom method that opens
            //stream for "myprotocol://"
            input = openMyInputStream(uri);
        } else {
            input = delegate.getInputStream(uri, info);
        }
        return input;
    }

    private InputStream openMyInputStream(String uri) {
        InputStream input = null;
        // @todo: open stream here
        return input;
    }
    public void releaseConnection(ConnectionInfo info)
        throws IOException, MediaException {
        Object o = info.getConnection();
        if (o instanceof MyProtocolConnection) {
            ((MyProtocolConnection)o).close(); // Perform cleanup.
        } else {
            delegate.releaseConnection(info);
        }
    }
    public void setProperty(String property, String value) {
        delegate.setProperty(property, value);
    }
    // Inner class that defines the connection class.
    public static class MyProtocolConnection {
        public MyProtocolConnection() {
            // ...
        }
        public void close() {
            // ...
        }
    }
}
```

3

Storing data

Use BlackBerry persistent storage
Manage persistent data
Manage custom objects
Use the MIDP record store

Use BlackBerry persistent storage

Storage method	Description
BlackBerry® persistence model	The BlackBerry persistence model provides a flexible and efficient way to store data. When writing an application specifically for BlackBerry devices, use the BlackBerry persistence model. • The BlackBerry persistence model lets you save any `Object` in the persistent store. As a result, searching for data in the persistent store is faster than searching in the record store model. To store custom object types, the class of the custom type must use the `Persistable` interface. • In the BlackBerry persistence model, applications can share data at the discretion of the application that creates the data. Code signing specifies that only authorized applications can access the data.
MIDP record stores	The MIDP record store allows an application to be portable across multiple devices that are compatible with the Java™ Platform, Micro Edition. • In MIDP, store persistent data as records in `RecordStore` objects. MIDP records store data only as byte arrays. • In MIDP, each `RecordStore` belongs to a single MIDlet suite. A MIDlet can access only the record stores that a MIDlet in the same suite creates.

BlackBerry persistent storage

Feature	Description
Security	By default, applications on the BlackBerry® device that are digitally signed by Research In Motion (RIM) can access the data in the persistent store. Contact RIM for information about controlling access to the data.
Administrative control	With the BlackBerry Enterprise Server, system administrators can use IT policies to control the use of persistent storage by third-party applications. Administrators can set ALLOW_USE_PERSISTENT_STORE to TRUE or FALSE. By default, third-party applications are enabled to use persistent storage (ALLOW_USE_PERSISTENT_STORE is TRUE). This policy does not affect the MIDP record store.

Feature	Description
Data integrity	To maintain the integrity of data in persistent storage, partial updates are not made if an error occurs during a commit. Data in the `PersistentObject` retains the values from the last commit in order to preserve data integrity. If the JVM performs an emergency garbage collection operation due to low memory, outstanding transactions are committed immediately to avoid compromising data integrity. If the BlackBerry device fails during this operation, partially completed transactions are committed when the BlackBerry device starts. Outstanding transactions are not committed during a normal garbage collection operation.

Manage persistent data

Task	Steps
Create a unique long key.	Each `PersistentObject` has a unique long key. 1. In the BlackBerry® Integrated Development Environment, type a string value, such as `com.rim.samples.docs.userinfo`. 2. Select this string. 3. Right-click this string and click **Convert 'com.rim.samples.docs.userinfo' to long**. 4. Include a comment in your code to indicate the string that you used to generate the unique long key.

Create a persistent data store.	1. Create a single static `PersistentObject`.
	`static PersistentObject store;`
	2. Invoke `PersistentStore.getPersistentObject`, using the unique long key as a parameter.
	`static {`
	`store = PersistentStore.getPersistentObject( 0xa1a569278238dad2L );}`

Store an object persistently.	1. Invoke `setContents()` on a `PersistentObject`. This method replaces existing content with the new content.
	2. To save the new content to the persistent store, invoke `commit()`.
	`String[] userinfo = {username, password};`
	`synchronized(store) {`
	`store.setContents(userinfo);`
	`store.commit();}`

Store objects in a batch transaction.	1. To use a batch transaction to commit objects to the persistent store, invoke `PersistentStore.getSynchObject()`. This method retrieves the persistent store monitor that locks the object. 2. Synchronize on the object. 3. Invoke `commit()` as necessary. If any commit in the batch fails, the entire batch transaction fails.
Commit a monitor object separately from a batch transaction.	> Invoke `forceCommit()` while synchronizing the monitor object.

Task	Steps

Retrieve persistent data.	1. Invoke `getContents()` on a `PersistentObject`.
	2. To convert the object that `PersistentObject.getContents()` returns to a specific object type, perform an explicit cast on the object that `PersistentObject.getContents()` returns.
	`synchronized(store) {`
	`String[] currentinfo = (String[])store.getContents();`
	`if(currentinfo == null) {`
	`Dialog.alert(_resources.getString(APP_ERROR));`
	`}`
	`else {`
	`currentusernamefield.setText(currentinfo[0]);`
	`currentpasswordfield.setText(currentinfo[1]);`
	`}}`

Delete all persistent data from an application.	If you delete the .cod file that defines a `PersistentStore`, then all persistent objects that the .cod file created are deleted. > Invoke `PersistentStore.destroyPersistentObject()`, providing as a parameter a unique key for the `PersistentObject`.
Delete specific persistent data from an application.	> To delete individual data, treat the data as normal object, and delete references to it. A garbage collected operation delete the data.

Code sample: Saving user name and password information

This code sample demonstrates how to create an application for BlackBerry® device users that lets them view their current user names and passwords, type new user names and passwords, and save changes.

Example: UserInfo.java

```
/**
 * UserInfo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.userinfo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.util.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

public class UserInfo extends UiApplication implements UserInfoResource
{
    private static PersistentObject store;
    private static ResourceBundle _resources;
    private AutoTextEditField usernamefield;
    private PasswordEditField passwordfield;
```

```
    private AutoTextEditField currentusernamefield;
    private AutoTextEditField currentpasswordfield;

    static {
        _resources = ResourceBundle.getBundle(
            UserInfoResource.BUNDLE_ID, UserInfoResource.BUNDLE_NAME);
        store = PersistentStore.getPersistentObject(0xa1a569278238dad2L);
    }

   private MenuItem saveItem = new MenuItem( _resources.getString(MENUITEM_SAVE), 110, 10)
{
        public void run() {
            String username = usernamefield.getText();
            String password = passwordfield.getText();
            String[] userinfo = {username, password};
            synchronized(store) {
                store.setContents(userinfo);
                store.commit();
            }

            Dialog.inform(_resources.getString(APP_SUCCESS));

            usernamefield.setText(null);
            passwordfield.setText(null);
        }
    };

    private MenuItem getItem = new MenuItem( _resources.getString(MENUITEM_GET), 110, 11 )
{
        public void run() {
            synchronized(store) {
                String[] currentinfo = (String[])store.getContents();
                if(currentinfo == null) {
                    Dialog.alert(_resources.getString(APP_ERROR));
                } else {
                    currentusernamefield.setText(currentinfo[0]);
                    currentpasswordfield.setText(currentinfo[1]);
                }
            }
        }
    };

    public static void main(String[] args) {
        UserInfo app = new UserInfo();
        app.enterEventDispatcher();
    }

    public UserInfo() {
        MainScreen mainScreen = new UserMainScreen();
        mainScreen.setTitle(new LabelField(
                _resources.getString(APPLICATION_TITLE)));

        usernamefield = new AutoTextEditField(
                _resources.getString(FIELD_NAME), "");
        passwordfield = new PasswordEditField(
            _resources.getString(FIELD_PASSWORD), "");
        currentusernamefield = new AutoTextEditField(
            _resources.getString(FIELD_CURRENTNAME), "");
```

```
        currentpasswordfield = new AutoTextEditField(
            _resources.getString(FIELD_CURRENTPASSWORD), “”);

        SeparatorField separator = new SeparatorField();

        mainScreen.add(usernamefield);
        mainScreen.add(passwordfield);
        mainScreen.add(separator);
        mainScreen.add(currentusernamefield);
        mainScreen.add(currentpasswordfield);
        pushScreen(mainScreen);
    }

    private final class UserMainScreen extends MainScreen
    {
        protected void makeMenu( Menu menu, int instance ) {
            menu.add(saveItem);
            menu.add(getItem);
            super.makeMenu(menu, 0);
        }

        public void close() {
            Dialog.alert(_resources.getString(APP_EXIT));
            super.close();
        }
    }
}
```

Manage custom objects

Task	Steps

Create an object to store data.	1. Create a `Vector` object in which to store multiple objects.
	`private static Vector _data;`
	2. Create a single static `PersistentObject`.
	`PersistentObject store;`
	`static {`
	`store = PersistentStore.getPersistentObject( 0xdec6a67096f833cL );`
	`//key is hash of test.samples.restaurants`
	`_data = (Vector)store.getContents();`
	`synchronized (store) {`
	`if (_data == null) {`
	`_data = new Vector();`
	`store.setContents(_data);`
	`store.commit();`
	`}`
	`}`
	`}`

Task	Steps

Store data persistently.	> In the class for the objects that you want to store, implement the `Persistable` interface.
	`private static final class RestaurantInfo implements Persistable {`
	`private String[] _elements;`
	`public static final int NAME = 0;`
	`public static final int ADDRESS = 1;`
	`public static final int PHONE  = 2;`
	`public static final int SPECIALTY = 3;`
	
	`public RestaurantInfo() {`
	`_elements = new String[4];`
	`for ( int i = 0; i < _elements.length; ++i) {`
	`_elements[i] = new String("");`
	`}`
	`}`
	
	`public String getElement(int id) {`
	`return _elements[id];`
	`}`
	`public void setElement(int id, String value) {`
	`_elements[id] = value;`
	`}`
	`}`

Save an object.	1. Define an object.
	The following code fragment creates a `RestaurantInfo` object and uses its set methods to define its components.
	`RestaurantInfo info = new RestaurantInfo();`
	`info.setElement(RestaurantInfo.NAME, namefield.getText());`
	`info.setElement(RestaurantInfo.ADDRESS,addressfield.getText());`
	`info.setElement(RestaurantInfo.PHONE, phonefield.getText());`
	`info.setElement(RestaurantInfo.SPECIALTY, specialtyfield.getText());`
	2. Add the object to a vector object by invoking `addElement()`.
	`_data.addElement(info);`
	3. Invoke `setContents()`.
	4. To save the updated data, invoke `commit()` on the `PersistentObject`.
	5. Synchronize with the persistent object when making changes so that other threads cannot make changes to the object at the same time.
	`synchronized(store) {`
	`store.setContents(_data);`
	`store.commit();`
	`}`

Task	Steps

Retrieve the most recently saved object.	> Invoke `_data.lastElement()`.
	`public void run() {`
	`synchronized(store) {`
	`_data = (Vector)store.getContents();`
	`if (!_data.isEmpty()) {`
	`RestaurantInfo info = (RestaurantInfo)_data.lastElement();`
	`namefield.setText(info.getElement(RestaurantInfo.NAME));`
	`addressfield.setText(info.getElement(RestaurantInfo.ADDRESS));`
	`phonefield.setText(info.getElement(RestaurantInfo.PHONE));`
	`specialtyfield.setText(info.getElement(`
	`RestaurantInfo.SPECIALTY));}`
	`}`
	`}`

Code sample: Storing and viewing restaurant information

This code sample lets BlackBerry® device users save information about multiple restaurants and view information about the most recently saved restaurant.

Example: Restaurants.java

```
 /**
 * Restaurants.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.restaurants;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.util.*;
import net.rim.device.api.i18n.*;
import net.rim.blackberry.api.invoke.*;
import net.rim.blackberry.api.browser.*;
import com.rim.samples.docs.resource.*;

public class Restaurants extends UiApplication implements RestaurantResource {

    private AutoTextEditField namefield;
    private AutoTextEditField addressfield;
    private EditField phonefield;
    private EditField websitefield;
    private EditField specialtyfield;

    private static Vector _data;
    private static PersistentObject store;
    private static ResourceBundle _resources;

    private MenuItem saveItem = new MenuItem(_resources.getString(MENUITEM_SAVE), 110, 10)
{
```

```
        public void run() {
            RestaurantInfo info = new RestaurantInfo();

            info.setElement(RestaurantInfo.NAME, namefield.getText());
            info.setElement(RestaurantInfo.ADDRESS, addressfield.getText());
            info.setElement(RestaurantInfo.PHONE, phonefield.getText());
            info.setElement(RestaurantInfo.WEBSITE, phonefield.getText());
            info.setElement(RestaurantInfo.SPECIALTY,
            specialtyfield.getText());

            _data.addElement(info);

            synchronized(store) {
                store.setContents(_data);
                store.commit();
            }
            Dialog.inform(_resources.getString(APP_SUCCESS));
            namefield.setText(null);
            addressfield.setText(null);
            phonefield.setText("");
            websitefield.setText("");
            specialtyfield.setText("");
        }
    };

    private MenuItem getItem = new MenuItem(_resources.getString(MENUITEM_GET), 110, 11) {
        public void run() {
            synchronized(store) {
                _data = (Vector)store.getContents();
                if (!_data.isEmpty()) {
                    RestaurantInfo info = (RestaurantInfo)_data.lastElement();
                    namefield.setText(info.getElement(RestaurantInfo.NAME));
                    addressfield.setText(info.getElement(RestaurantInfo.ADDRESS));
                    phonefield.setText(info.getElement(RestaurantInfo.PHONE));
                    websitefield.setText(info.getElement(RestaurantInfo.WEBSITE));
                    specialtyfield.setText(info.getElement(RestaurantInfo.SPECIALTY));
                }
            }
        }
    };

    private MenuItem phoneItem = new MenuItem(_resources.getString(MENUITEM_PHONE), 110,
12) {
        public void run() {
            synchronized(store) {
                String phoneNumber = phonefield.getText();
                if ( phoneNumber.length() == 0) {
                    Dialog.alert(_resources.getString(ALERT_NO_PHONENUMBER));
                } else {
                    PhoneArguments call = new PhoneArguments(PhoneArguments.ARG_CALL,
phoneNumber);
                    Invoke.invokeApplication(Invoke.APP_TYPE_PHONE, call);
                }
            }
        }
    };
    private MenuItem browserItem = new MenuItem(_resources.getString(MENUITEM_BROWSER),
110, 12) {
```

```
    public void run() {
        synchronized(store) {
            String websiteUrl = websitefield.getText();
            if (websiteUrl.length() == 0) {
                Dialog.alert(_resources.getString(ALERT_NO_WEBSITE));
            } else {
                BrowserSession visit = Browser.getDefaultSession();
                visit.displayPage(websiteUrl);
            }
        }
    }
};
static {
    _resources = ResourceBundle.getBundle(
    RestaurantResource.BUNDLE_ID,
    RestaurantResource.BUNDLE_NAME);
    store = PersistentStore.getPersistentObject(0xdec6a67096f833cL);
    // Key is hash of test.samples.restaurants.
    synchronized (store) {
        _data = (Vector)store.getContents();
        if (_data == null) {
            _data = new Vector();
            store.setContents( _data );
            store.commit();
        }
    }

}

public static void main(String[] args) {
    Restaurants app = new Restaurants();
    app.enterEventDispatcher();
}

private static final class RestaurantInfo implements Persistable {

    // Data.
    private String[] _elements;

    // Fields.
    public static final int NAME = 0;
    public static final int ADDRESS = 1;
    public static final int PHONE = 2;
    public static final int WEBSITE = 3;
    public static final int SPECIALTY = 4;

    public RestaurantInfo() {
        _elements = new String[4];
        for ( int i = 0; i < _elements.length; ++i) {
            _elements[i] = "";
        }
    }

    public String getElement(int id) {
        return _elements[id];
    }

    public void setElement(int id, String value) {
```

```
            _elements[id] = value;
        }
    }

    private final class RestaurantsMainScreen extends MainScreen
    {
        protected void makeMenu( Menu menu, int instance ) {
            menu.add(saveItem);
            menu.add(getItem);
            menu.add(phoneItem);
            menu.add(browserItem);
            super.makeMenu(menu, instance);
        }

        public void close() {
            Dialog.alert(_resources.getString(APP_EXIT));
            super.close();
        }
    }

    public Restaurants() {
        MainScreen mainScreen = new RestaurantsMainScreen();
        mainScreen.setTitle(new LabelField(
        _resources.getString(APPLICATION_TITLE)));
        namefield = new AutoTextEditField(
        _resources.getString(FIELD_NAME), "");
        addressfield = new AutoTextEditField(
        _resources.getString(FIELD_ADDRESS), "");
        phonefield = new EditField(
        _resources.getString(FIELD_PHONE), "", Integer.MAX_VALUE,
        BasicEditField.FILTER_PHONE);
        websitefield = new EditField(
        _resources.getString(FIELD_WEBSITE), "", Integer.MAX_VALUE,
        BasicEditField.FILTER_URL);
        specialtyfield = new EditField(
        _resources.getString(FIELD_SPECIALTY), "",
        Integer.MAX_VALUE, BasicEditField.FILTER_DEFAULT);

        mainScreen.add(namefield);
        mainScreen.add(addressfield);
        mainScreen.add(phonefield);
        mainScreen.add(websitefield);
        mainScreen.add(specialtyfield);
        pushScreen(mainScreen);
    }
}
```

Use the MIDP record store

Task	Steps
Create a record store.	> Invoke `openRecordStore()` and specify true to indicate that the method should create the record store if the record store does not exist. `RecordStore store = RecordStore.openRecordStore("Contacts", true);`
Add a record.	> Invoke `addRecord()`. `int id = store.addRecord(_data.getBytes(), 0, _data.length());`

Retrieve a record.	> Invoke `getRecord(int, byte[], int)` and provide the following parameters:
	• a record ID
	• a byte array
	• an offset
	`byte[] data = new byte[store.getRecordSize(id)];`
	`store.getRecord(id, data, 0);`
	`String dataString = new String(data);`

Retrieve all records.	1. Invoke `openRecordStore()`.
	`RecordStore store = RecordStore.openRecordStore("Contacts", false);`
	2. Invoke `enumerateRecords()` with the following parameters:
	• `filter`: specifies a `RecordFilter` object to retrieve a subset of record store records (if null, the method returns all records)
	• `comparator`: specifies a `RecordComparator` object to determine the order in which the method returns the records (if null, the method returns the records in any order)
	• `keepUpdated`: determines if the method keeps the enumeration current with the changes to the record store
	`RecordEnumeration e = store.enumerateRecords(null, null, false);`

4

Managing data

Data synchronization
Backing up and restoring data

Data synchronization

Research In Motion (RIM) does not provide tools or applications for synchronizing data to remote data sources, so you must build the synchronization logic into your application. See the *BlackBerry Java Development Environment Fundamentals Guide* for more information about creating applications for sychronizing data on a BlackBerry® device.

Types of data synchronization

Synchronization type	Description
Wireless (BlackBerry® Enterprise Server)	The automatic wireless backup process on a BlackBerry Enterprise Server is designed to back up data from the BlackBerry device to the BlackBerry Enterprise Server. By default, wireless backup is active on the BlackBerry Enterprise Server. See the *BlackBerry Enterprise Server for Microsoft Exchange Feature and Technical Overview* for more information about the BlackBerry Enterprise Server. When the automatic wireless backup process runs on the BlackBerry Enterprise Server, the process saves application data with the user account settings and the other BlackBerry device data that backs up.
Wireless (XML data)	An application uses XML APIs to generate and parse XML-formatted data to send and receive over a wireless connection.
Desktop-based (BlackBerry Desktop Manager)	An application uses a USB connection to a computer to synchronize data with a desktop application. This type of synchronization requires the use of the BlackBerry Desktop Synchronization APIs, the BlackBerry Desktop Manager, and a desktop application that can read data from the BlackBerry device using the BlackBerry Desktop Manager adapter. A BlackBerry device user must manually start the synchronization process by running the BlackBerry Desktop Manager, which notifies the application on the BlackBerry device to send the data to the desktop application.
Desktop-based (USB protocols)	An application uses a USB connection to a computer and native USB protocols to synchronize data with a desktop application.

Backing up and restoring data

Add support for backing up data over the wireless network

Task	Steps
Set up the BlackBerry® Enterprise Server to back up the application data using automatic wireless backup.	> Implement the `OTASyncCapable` and `CollectionEventSource` interfaces.

Activate the synchronization process when the BlackBerry device starts.	> In the main method, create code that activates the synchronization process.
	`public static void main(String[] args) {`
	`boolean startup = false;`
	`for (int i=0; i<args.length; ++i) {`
	`if (args[i].startsWith("init")) {`
	`startup = true;`
	`}`
	`}`
	`if (startup) {`
	`//enable application for synchronization on startup`
	`SerialSyncManager.getInstance().enableSynchronization(new RestaurantsSync());`
	`} else {`
	`RestaurantsSync app = new RestaurantsSync();`
	`app.enterEventDispatcher();`
	`}`
	`}`
	The first time the BlackBerry device starts, the **Alternate CLDC Application Entry Point** project passes an argument to the application so that the application registers only once.

Create a project that acts as an alternate entry point to the main application.	MIDlet applications do not support this task. 1. In the BlackBerry Integrated Development Environment, create a project. 2. Right-click the project, and then click **Properties**. 3. Click the **Application** tab. 4. In the **Project type** drop-down list, click **Alternate CLDC Application Entry Point**. 5. In the **Alternate entry point for** drop-down list, click the project that starts the synchronization process. 6. In the **Arguments passed to** field, type **init**. Make sure the value you type in the **Arguments passed to** field matches the value in the **startsWith** argument in the main method of your application. 7. Select the **Auto-run on startup** option. 8. Select the **System module** option. 9. Click **OK**.

Provide an application with schema data for a SyncCollection.	> In your implementation of the `OTASyncCapable` interface, implement the `getSchema()` method.
	`public SyncCollectionSchema getSchema()`
	`{// returns our schema`
	`return _schema;`
	`}`

Task	Steps

Uniquely identify each record type in a SyncCollection.	> Invoke the `SyncCollectionSchema.setDefaultRecordType()` method. The following example shows only one record type, so it uses the default record type:
	`private static final int DEFAULT_RECORD_TYPE = 1;`
	`_schema = new SyncCollectionSchema();`
	`_schema.setDefaultRecordType(DEFAULT_RECORD_TYPE);`

Uniquely identify each record in a SyncCollection.	> Invoke the `SyncCollectionSchema.setKeyFieldIDs()` method.
	`private static final int[] KEY_FIELD_IDS = new int[] {FIELDTAG_FIRST_NAME, FIELDTAG_LAST_NAME};`
	`_schema.setKeyFieldIds(DEFAULT_RECORD_TYPE, KEY_FIELD_IDS);`

Access a SyncCollection

Task	Steps

Retrieve an instance of the SyncCollection from the RunTimeStore.	> To make sure that the application works with only one version of the `SyncCollection`, implement a static method that returns an instance of the `SyncCollection`.
	`static OTABackupRestoreContactCollection getInstance()`
	`{`
	`RuntimeStore rs = RuntimeStore.getRuntimeStore();`
	`synchronized( rs )`
	`{`
	`OTABackupRestoreContactCollection collection = (OTABackupRestoreContactCollection)rs.get( AR_KEY );`
	`if( collection == null )`
	`{`
	`collection = new OTABackupRestoreContactCollection();`
	`rs.put( AR_KEY, collection );`
	`}`
	`return collection;`
	`}`
	`}`

Retrieve the SyncCollection from the PersistentStore.	1. To provide the application with access to the newest `SyncCollection` data from the `PersistentStore`, invoke the `PersistentStore.getPersistentObject()` method using the ID of the `SyncCollection`.
	2. Store the returned data in a vector object.
	`private PersistentObject _persist;  // The persistable object for the contacts.`
	`private Vector _contacts;  // The actual contacts.`
	`private static final long PERSISTENT_KEY = 0x266babf899b20b56L;`
	`_persist = PersistentStore.getPersistentObject( PERSISTENT_KEY );`
	`_contacts = (Vector)_persist.getContents();`
	3. Create a method to provide the application with the newest `SyncCollection data` before a wireless data backup session begins.
	`public void beginTransaction()`
	`{`
	`_persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);`
	`_contacts = (Vector)_persist.getContents();`
	`}`
	4. Create code to manage the case where the `SyncCollection` you retrieve from the `PersistentStore` is empty.
	`if( _contacts == null )`
	`{`
	`_contacts = new Vector();`
	`_persist.setContents( _contacts );`
	`_persist.commit();`
	`}`

Notify the system when a SyncCollection changes

Task	Steps

Use a collection listener to notify the system when a `SyncCollection` changes.	The system invokes `CollectionEventSource.addCollectionListener()` to create a `CollectionListener` for each `SyncCollection` that the application makes available for wireless backup.
	1. Create a private vector object to store the collection of `SyncCollection` listeners for the application.
	`private Vector _listeners;`
	`_listeners = new CloneableVector();`
	2. Implement the `CollectionListener()` method, making sure that the method adds a `CollectionListener` to the vector object.
	`public void addCollectionListener(Object listener)`
	`{`
	`_listeners = ListenerUtilities.fastAddListener( _listeners, listener );`
	`}`

Remove a collection listener.	When the system no longer requires a `CollectionListener`, it invokes `removeCollectionListener`.
	> Implement the `removeCollectionListener()` method, using the `ListenerUtilities.removeListener()` method to remove a `CollectionListener` from the collection of `SyncCollection` listeners for the application.
	`public void removeCollectionListener(Object listener)`
	`{_listeners = ListenerUtilities.removeListener( _listeners, listener );`
	`}`

Notify the system when an element is added to a SyncCollection.	> Invoke `CollectionListener.elementAdded()`:
	`for( int i=0; i<_listeners.size(); i++ )`
	`{`
	`CollectionListener cl = (CollectionListener)_listeners.elementAt( i );`
	`cl.elementAdded( this, object );`
	`}`
	`return true;`
	`}`

Notify the system when an element is removed from a SyncCollection.	> Invoke `CollectionListener.elementRemoved()`.
Notify the system when an element in a SyncCollection is replaced.	> Invoke `CollectionListener.elementUpdated()`.

Using SyncObjects

Task	Steps

Retrieve SyncObjects from the SyncCollection.	> Implement the `getSyncObjects()` method.
	`public SyncObject[] getSyncObjects()`
	`{//Retrieve the contact data.`
	`SyncObject[] contactArray = new SyncObject[_contacts.size()];`
	`for (int i = _contacts.size() - 1; i >= 0; --i)`
	`{`
	`contactArray[i] = (SyncObject)_contacts.elementAt(i);`
	`}`
	`return contactArray;`
	`}`

Access a specific SyncObject.	> Implement the `getSyncObject()` method, using the **_uid** variable to retrieve a specific `SyncObject`.
	`public SyncObject getSyncObject(int uid)`
	`{`
	`for (int i = _contacts.size() - 1; i >= 0; --i)`
	`{`
	`SyncObject so = (SyncObject)_contacts.elementAt(i);`
	`if ( so.getUID() == uid ) return so;`
	`}`
	`return null;`
	`}`

Add a SyncObject to the SyncCollection.	> Create a method that adds `SyncObjects` to the `PersistentStore` object.
	`public boolean addSyncObject(SyncObject object)`
	`{`
	`// Add a contact to the PersistentStore object.`
	`_contacts.addElement(object);`
	`}`

Save a SyncCollection.	Before a wireless backup session ends, save the newest `SyncCollection` data.
	`Invoke the setContents() and commit()methods.`
	`public void endTransaction()`
	`{`
	`_persist.setContents(_contacts);`
	`_persist.commit();`
	`}`

Code sample: Using a SyncCollection to back up data over the wireless network

Example: OTABackupRestoreContactCollection.java

```
/*
 * OTABackupRestoreContactCollection.java
 *
 * AUTO_COPYRIGHT_SUB_TAG
 */

package com.rim.samples.device.otabackuprestoredemo;
```

```
import java.io.*;
import java.util.*;
import net.rim.device.api.collection.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.synchronization.*;
import net.rim.device.api.util.*;
import net.rim.device.api.system.*;


/**
 * A collection enabled for OTA backup/restore. Basically a serially syncable collection
 * with few added interfaces.
 */
class OTABackupRestoreContactCollection implements SyncConverter, SyncCollection,
        OTASyncCapable, CollectionEventSource
{
    private static final long PERSISTENT_KEY = 0x266babf899b20b56L; //
com.rim.samples.device.otabackuprestoredemo.OTABackupRestoreContactCollection._persist
    private static final long AR_KEY         = 0xef780e08b3a7cf07L; //
com.rim.samples.device.otabackuprestoredemo.OTABackupRestoreContactCollection

    private PersistentObject _persist;      // the persistable object for the contacts
    private Vector _contacts;               // the actual contacts
    private Vector _listeners;              // listeners to generate events when contacts
are added
    private SyncCollectionSchema _schema;   // lets us know about the data we are backing
up

    private static final int FIELDTAG_FIRST_NAME = 1;
    private static final int FIELDTAG_LAST_NAME = 2;
    private static final int FIELDTAG_EMAIL_ADDRESS = 3;

    private static final int DEFAULT_RECORD_TYPE = 1;       // the id for the default (and
the only) record type
    private static final int[] KEY_FIELD_IDS = new int[] {  // key fields - lets the server
know which fields uniquely define a record
        FIELDTAG_FIRST_NAME,
        FIELDTAG_LAST_NAME
    };


    private OTABackupRestoreContactCollection()
    {
        _persist = PersistentStore.getPersistentObject( PERSISTENT_KEY );
        _contacts = (Vector)_persist.getContents();

        if( _contacts == null )
        {
            _contacts = new Vector();
            _persist.setContents( _contacts );
            _persist.commit();
        }

        _listeners = new CloneableVector();

        // set up the schema for the collection
        _schema = new SyncCollectionSchema();
```

```
        _schema.setDefaultRecordType(DEFAULT_RECORD_TYPE);
        _schema.setKeyFieldIds(DEFAULT_RECORD_TYPE, KEY_FIELD_IDS);
    }

    static OTABackupRestoreContactCollection getInstance()
    {
        RuntimeStore rs = RuntimeStore.getRuntimeStore();
        synchronized( rs )
        {
            OTABackupRestoreContactCollection collection =
(OTABackupRestoreContactCollection)rs.get( AR_KEY );
            if( collection == null )
            {
                collection = new OTABackupRestoreContactCollection();
                rs.put( AR_KEY, collection );
            }
            return collection;
        }
    }

    //SyncConverter methods----------
    public boolean convert(SyncObject object, DataBuffer buffer, int version)
    {
        if (version == getSyncVersion())
        {
            if (object instanceof ContactData)
            {
                String first = ((ContactData)object).getFirst();
                String last = ((ContactData)object).getLast();
                String email = ((ContactData)object).getEmail();

                //in compliance with desktop sync format
                buffer.writeShort(first.length()+1);
                buffer.writeByte(FIELDTAG_FIRST_NAME);
                buffer.write(first.getBytes());
                buffer.writeByte(0);
                buffer.writeShort(last.length()+1);
                buffer.writeByte(FIELDTAG_LAST_NAME);
                buffer.write(last.getBytes());
                buffer.writeByte(0);
                buffer.writeShort(email.length()+1);
                buffer.writeByte(FIELDTAG_EMAIL_ADDRESS);
                buffer.write(email.getBytes());
                buffer.writeByte(0);


                return true;
            }
        }
        return false;
    }

    public SyncObject convert(DataBuffer data, int version, int UID)
    {
        try {
            ContactData contact = new ContactData(UID);
            while(data.available() > 0)
            {
```

```
                int length = data.readShort();
                byte[] bytes = new byte[length];
                switch (data.readByte())
                {
                    case FIELDTAG_FIRST_NAME:
                        data.readFully(bytes);
                        //trim null-terminator
                        contact.setFirst(new String(bytes).trim());
                        break;
                    case FIELDTAG_LAST_NAME:
                        data.readFully(bytes);
                        contact.setLast(new String(bytes).trim());
                        break;
                    case FIELDTAG_EMAIL_ADDRESS:
                        data.readFully(bytes);
                        contact.setEmail(new String(bytes).trim());
                        break;
                    default:
                        data.readFully(bytes);
                        //other fields not supported
                        break;
                }
            }
            return contact;
        }
        catch (EOFException e)
        {
            System.err.println(e.toString());
        }
        return null;
    }

    //SyncCollection methods----------
    public boolean addSyncObject(SyncObject object)
    {
        // add a contact to the persistent store
        _contacts.addElement(object);
        _persist.setContents( _contacts );
        _persist.commit();

        // we want to let any collection listeners we have that the collection has been
changed
        for( int i=0; i<_listeners.size(); i++ )
        {
            CollectionListener cl = (CollectionListener)_listeners.elementAt( i );
            cl.elementAdded( this, object );
        }
        return true;
    }

    public boolean updateSyncObject(SyncObject oldObject, SyncObject newObject)
    {
        return false; //na - this method would look much the same as addSyncObject
    }

    public boolean removeSyncObject(SyncObject object)
    {
        return false; //na - this method would look much the same as addSyncObject
```

```
    }

    public boolean removeAllSyncObjects()
    {
        return false; //na
    }

    public SyncObject[] getSyncObjects()
    {
        SyncObject[] contactArray = new SyncObject[_contacts.size()];
        for (int i = _contacts.size() - 1; i >= 0; --i)
        {
            contactArray[i] = (SyncObject)_contacts.elementAt(i);
        }
        return contactArray;
    }

    public SyncObject getSyncObject(int uid)
    {
        for (int i = _contacts.size() - 1; i >= 0; --i)
        {
            SyncObject so = (SyncObject)_contacts.elementAt(i);
            if ( so.getUID() == uid ) return so;
        }
        return null;
    }

    public boolean isSyncObjectDirty(SyncObject object)
    {
        return false; //na
    }

    public void setSyncObjectDirty(SyncObject object)
    {
        //na
    }

    public void clearSyncObjectDirty(SyncObject object)
    {
        //na
    }

    public int getSyncObjectCount()
    {
        _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
        _contacts = (Vector)_persist.getContents();
        return _contacts.size();
    }

    public int getSyncVersion()
    {
        return 1;
    }

    public String getSyncName()
    {
        return “OTABackupRestoreContacts”;
    }
```

```
    public String getSyncName(Locale locale)
    {
        return null;
    }

    public SyncConverter getSyncConverter()
    {
        return this;
    }

    public void beginTransaction()
    {
        _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
        _contacts = (Vector)_persist.getContents();
    }

    public void endTransaction()
    {
        _persist.setContents(_contacts);
        _persist.commit();
    }

    //OTASyncCapable methods ------------------------------------------------
    public SyncCollectionSchema getSchema()
    {
        // returns our schema
        return _schema;
    }

    //CollectionEventSource methods -----------------------------------------
    public void addCollectionListener(Object listener)
    {
        _listeners = ListenerUtilities.fastAddListener( _listeners, listener );
    }

    public void removeCollectionListener(Object listener)
    {
        _listeners = ListenerUtilities.removeListener( _listeners, listener );
    }

    public int size()
    {
        return _contacts.size();
    }

    public ContactData contactAt( int index )
    {
        return (ContactData)_contacts.elementAt( index );
    }
}
```

Add support for backing up data with the BlackBerry Desktop Software

Task	Steps
Let your application maintain a collection of synchronized objects, producing and processing valid synchronization data when creating a SyncObject.	Implement the `SyncCollection` and `SyncConverter` interfaces by the same class or by separate classes, depending on the design of the application. > Change the main class for the application to implement the `SyncCollection` and `SyncConverter` interfaces. `public class RestaurantsSync extends UiApplication implements RestaurantsSyncResource,SyncCollection, SyncConverter`
Let persistable objects be synchronization objects.	> Modifiy a class that implements the `Persistable` interface to implement the `SyncObject` interface. `private static final class RestaurantInfo implements Persistable, SyncObject`
Create a unique ID for a synchronization object.	> In the persistable class, create an instance variable for storing a unique ID for synchronization operations. `private int _uid;`

Let your main application retrieve the unique ID of the synchronization object.	> In the persistable class, implement the `getUID()` method to return a unique ID for synchronization operations.
	`public int getUID() {`
	`return _uid;`
	`}`

Let your main application create a synchronization object using a unique ID.	> In the persistable class, create a constructor that accepts a unique ID as a parameter and sets the **_uid** variable to this value.
	`public RestaurantInfo(int uid) {`
	`_elements = new String[4];`
	`for (int i = 0; i < _elements.length; ++i) {`
	`_elements[i] = "";`
	`}`
	`_uid = uid;`
	`}`

Activate synchronization when the BlackBerry device starts

Task	Steps

Activate synchronization when the BlackBerry® device starts.	The first time the BlackBerry device starts, the **Alternate CLDC Application Entry Point** project passes an argument to the application so that the application registers only once.
	> In the main method of the application, create code that activates the synchronization process.
	`public static void main(String[] args) {`
	`boolean startup = false;`
	`for (int i=0; i<args.length; ++i) {`
	`if (args[i].startsWith("init")) {`
	`startup = true;`
	`}`
	`}`
	`if (startup) {`
	`//enable application for synchronization on startup`
	`SerialSyncManager.getInstance().enableSynchronization(new RestaurantsSync());`
	`} else {`
	`RestaurantsSync app = new RestaurantsSync();`
	`app.enterEventDispatcher();`
	`}`
	`}`

Create a project that acts as an alternate entry point to the main application.	If the application is a MIDlet, arguments cannot pass to the application when the BlackBerry device starts. 1. In the BlackBerry Integrated Development Environment, create a project. 2. Right-click the project, and then click **Properties**. 3. Click the **Application** tab. 4. In the **Project type** drop-down list, click **Alternate CLDC Application Entry Point**. 5. In the **Alternate entry point for** drop-down list, click the project that implements synchronization. 6. In the **Arguments passed to** field, type **init**. Make sure that the value you type in the **Arguments passed to** field matches the value in the **startsWith** argument in the main method of the application. 7. Select the **Auto-run on startup** option. 8. Select the **System module** option. 9. Click **OK**.

Code sample: Letting the BlackBerry Desktop Software back up and restore application data

Example: RestaurantsSync.java

```
/**
 * RestaurantsSync.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
```

```
package com.rim.samples.docs.restaurantssync;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.util.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.synchronization.*;
import com.rim.samples.docs.resource.*;

public class RestaurantsSync extends UiApplication implements RestaurantsSyncResource,
        SyncCollection, SyncConverter {

    private static final long KEY = 0xdec6a67096f833cL;

    private AutoTextEditField namefield;
    private AutoTextEditField addressfield;
    private EditField phonefield;
    private EditField specialtyfield;

    private static PersistentObject store;
    private static Vector _data;
    private static ResourceBundle _resources;
    private static final int FIELDTAG_NAME = 1;
    private static final int FIELDTAG_PHONE = 2;
    private static final int FIELDTAG_ADDRESS = 3;
    private static final int FIELDTAG_SPECIALTY = 4;

    private static RestaurantsSync _instance;

    private MenuItem saveItem = new MenuItem(_resources, MENUITEM_SAVE, 110, 10) {
        public void run() {
            RestaurantInfo info = new RestaurantInfo();
            info.setElement(RestaurantInfo.NAME, namefield.getText());
            info.setElement(RestaurantInfo.ADDRESS, addressfield.getText());
            info.setElement(RestaurantInfo.PHONE, phonefield.getText());
            info.setElement(RestaurantInfo.SPECIALTY, specialtyfield.getText());
            _data.addElement(info);

            synchronized(store) {
                store.setContents(_data);
                store.commit();
            }
            Dialog.inform(_resources.getString(APP_SUCCESS));
            namefield.setText(null);
            addressfield.setText(null);
            phonefield.setText("");
            specialtyfield.setText("");
        }
    };

    private MenuItem getItem = new MenuItem(_resources, MENUITEM_GET, 110, 11) {
        public void run() {
            synchronized(store) {
```

```
            _data = (Vector)store.getContents();
            if (!_data.isEmpty()) {
                RestaurantInfo info = (RestaurantInfo)_data.lastElement();
                namefield.setText(info.getElement(RestaurantInfo.NAME));
                addressfield.setText(info.getElement(RestaurantInfo.ADDRESS));
                phonefield.setText(info.getElement(RestaurantInfo.PHONE));
                specialtyfield.setText(info.getElement(
                        RestaurantInfo.SPECIALTY));
            }
        }
    }
};

static {
    _resources = ResourceBundle.getBundle(RestaurantsSyncResource.BUNDLE_ID,
            RestaurantsSyncResource.BUNDLE_NAME);
    store = PersistentStore.getPersistentObject(KEY);
    synchronized (store) {
        _data = (Vector)store.getContents();
        if ( _data == null ) {
            _data = new Vector();
            store.setContents( _data );
            store.commit();
        }
    }

}

public static void main(String[] args) {
    boolean startup = false;
    for (int i=0; i<args.length; ++i) {
        if (args[i].startsWith("init")) {
            startup = true;
        }
    }

    if (startup) {
        // Enable application for synchronization on startup.
        SyncManager.getInstance().enableSynchronization(
                RestaurantsSync.getInstance());
    } else {
        RestaurantsSync app = new RestaurantsSync();
        app.enterEventDispatcher();
    }
}

public static RestaurantsSync getInstance() {
    if (_instance == null) {
        _instance = new RestaurantsSync();
    }
    return _instance;
}

private static final class RestaurantInfo implements Persistable, SyncObject {
    private String[] _elements; // Data.
    public static final int NAME = 0;
    public static final int ADDRESS = 1;
    public static final int PHONE  = 2;
```

```
        public static final int SPECIALTY = 3;
        private int _uid;

        public int getUID() {
            return _uid;
        }

        public RestaurantInfo() {
            _elements = new String[4];
            for ( int i = 0; i < _elements.length; ++i) {
                _elements[i] = "";
            }
        }
        public RestaurantInfo(int uid) {
            _elements = new String[4];
            for (int i = 0; i < _elements.length; ++i) {
                _elements[i] = "";
            }
            _uid = uid;
        }

        public String getElement(int id) {
            return _elements[id];
        }

        public void setElement(int id, String value) {
            _elements[id] = value;
        }
    }

    // SyncConverter methods.
    public SyncObject convert(DataBuffer data, int version, int UID) {
        try {
            RestaurantInfo info = new RestaurantInfo(UID);
            while(data.available() > 0) {
                int length = data.readShort();
                byte[] bytes = new byte[length];
                switch (data.readByte()) {
                    case FIELDTAG_NAME:
                        data.readFully(bytes);
                        //trim null-terminator
                        info.setElement(RestaurantInfo.NAME,
                                new String(bytes).trim());
                        break;
                    case FIELDTAG_PHONE:
                        data.readFully(bytes);
                        info.setElement(RestaurantInfo.PHONE,
                                new String(bytes).trim());
                        break;
                    case FIELDTAG_ADDRESS:
                        data.readFully(bytes);
                        info.setElement(RestaurantInfo.ADDRESS,
                                new String(bytes).trim());
                        break;
                    case FIELDTAG_SPECIALTY:
                        data.readFully(bytes);
                        info.setElement(RestaurantInfo.SPECIALTY,
                                new String(bytes).trim());
```

```
                    break;
                default:
                    data.readFully(bytes);
                    break;
            }
        }
        return info;
    } catch (EOFException e) {
        System.err.println(e.toString());
    }
    return null;
}

public boolean convert(SyncObject object, DataBuffer buffer, int version) {
    if (version == getSyncVersion()) {
        if (object instanceof RestaurantInfo )
        {
            String name = ((RestaurantInfo)object).getElement(
                    RestaurantInfo.NAME);
            String phone = ((RestaurantInfo)object).getElement(
                    RestaurantInfo.PHONE);
            String address = ((RestaurantInfo)object).getElement(
                    RestaurantInfo.ADDRESS);
            String specialty = ((RestaurantInfo)object).getElement(
                    RestaurantInfo.SPECIALTY);
            buffer.writeShort(name.length()+1);
            buffer.writeByte(FIELDTAG_NAME);
            buffer.write(name.getBytes());
            buffer.writeByte(0);
            buffer.writeShort(phone.length()+1);
            buffer.writeByte(FIELDTAG_PHONE);
            buffer.write(phone.getBytes());
            buffer.writeByte(0);
            buffer.writeShort(address.length()+1);
            buffer.writeByte(FIELDTAG_ADDRESS);
            buffer.write(address.getBytes());
            buffer.writeByte(0);
            buffer.writeShort(specialty.length()+1);
            buffer.writeByte(FIELDTAG_SPECIALTY);
            buffer.write(specialty.getBytes());
            buffer.writeByte(0);
            return true;
        }
    }
    return false;
}

public void beginTransaction() {
    store = PersistentStore.getPersistentObject(KEY);
    _data = (Vector)store.getContents();
}

public void endTransaction() {
    store.setContents(_data);
    store.commit();
}

public SyncConverter getSyncConverter() {
```

```
        return this;
    }

    public String getSyncName() {
        return “Restaurant Synchronization Demo”;
    }

    public String getSyncName(Locale locale) {
        return getSyncName();
    }

    public int getSyncObjectCount() {
        store = PersistentStore.getPersistentObject(KEY);
        _data = (Vector)store.getContents();
        return _data.size();
    }

    public SyncObject[] getSyncObjects() {
        SyncObject[] array = new SyncObject[_data.size()];
        for (int i = _data.size() - 1; i >= 0; --i) {
            array[i] = (SyncObject)_data.elementAt(i);
        }
        return array;
    }

    public SyncObject getSyncObject(int uid) {
        for (int i = _data.size() -1; i>= 0; --i) {
            SyncObject so = (SyncObject)_data.elementAt(i);
            if (so.getUID() == uid ) return so;
        }
        return null;
    }

    public int getSyncVersion() {
        return 1;
    }

    public boolean addSyncObject(SyncObject object) {
        _data.addElement(object);
        return true;
    }

    public boolean removeAllSyncObjects() {
        _data.removeAllElements();
        return true;
    }

    public void clearSyncObjectDirty(SyncObject object) {
        // Not applicable.
    }

    public boolean isSyncObjectDirty(SyncObject object) {
        return false;
    }

    public boolean removeSyncObject(SyncObject object) {
        return false;
    }
```

```
    public void setSyncObjectDirty(SyncObject object) {
    }

    public boolean updateSyncObject(SyncObject oldObject, SyncObject newObject) {
        return false;
    }

    public RestaurantsSync() {
        MainScreen mainScreen = new RestaurantsMainScreen();
        mainScreen.setTitle(new LabelField( _resources.getString(APPLICATION_TITLE)));
        namefield = new AutoTextEditField(_resources.getString(FIELD_NAME), "");
        addressfield = new AutoTextEditField( _resources.getString(FIELD_ADDRESS), "");
        phonefield = new EditField(                    _resources.getString(FIELD_PHONE),
"", Integer.MAX_VALUE, BasicEditField.FILTER_PHONE);
        specialtyfield = new EditField(_resources.getString(FIELD_SPECIALTY), "",
Integer.MAX_VALUE, BasicEditField.FILTER_DEFAULT);
        mainScreen.add(namefield);
        mainScreen.add(addressfield);
        mainScreen.add(phonefield);
        mainScreen.add(specialtyfield);
        pushScreen(mainScreen);
    }

    private final class RestaurantsMainScreen extends MainScreen
    {
        protected void makeMenu( Menu menu, int instance ) {
            menu.add(saveItem);
            menu.add(getItem);
            super.makeMenu(menu, instance);
        }

        public void close() {
            Dialog.alert(_resources.getString(APP_EXIT));
            super.close();
        }
    }
}
```

5

Using smart cards

Smart cards
Creating a smart card driver

Smart cards

Adding support for unsupported smart cards

Smart cards are credit card-sized devices that are designed to store and transfer sensitive data.

Smart card scenario	Description
Working with supported smart cards.	The BlackBerry® device supports the following smart cards: • CAC • SafeNet smart card (Datakey Model 330)
Working with unsupported smart cards.	If your smart card is not a CAC or SafeNet smart card, use the smart card API to write a BlackBerry device driver to support your smart card. The smart card API provides a library of components, in the `net.rim.device.api.smartcard package`, for interacting with smart cards and smart card readers. Drivers that correctly implement the smart card API can work with the S/MIME application on an S/MIME-enabled BlackBerry device. They can import certificates from the smart card and perform private key operations on the smart card, such as sign and decrypt messages. See the *BlackBerry with the S/MIME Support Package* for more information about S/MIME.

Creating a smart card driver

To create a smart card driver, complete the following tasks:

1. Create a smart card object.
2. Enable `libMain()` on startup.
3. Create a session for the smart card driver.
4. Enable a `CryptoToken` for private key operations.
5. Store the private key file location.

Create a smart card object

Task	Steps
Register a smart card driver with the `SmartCardFactory`.	> Extend the abstract `CryptoSmartCard` class.

Task	Steps
Let a smart card driver to open a session with a smart card.	> Implement `SmartCard.openSessionImpl(SmartCardReaderSession)`.
Let a smart card driver to verify the compatibility of a smart card with a specific ATR.	> Implement `SmartCard.checkAnswerToResetImpl(AnswerToReset)`.
Let a smart card driver to display settings or properties.	> Implement `SmartCard.displaySettingsImpl(Object)`.
Let a smart card driver to indicate support for display settings.	> Implement `SmartCard.isDisplaySettingsAvailableImpl(Object)`.
Retrieve the capabilities of a smart card.	> Implement `SmartCard.getCapabilitiesImpl()`.
Determine the smart card type.	> Implement `SmartCard.getLabelImpl()`.
Retrieve a string representation of the smart card.	> Implement `SmartCard.toString()`.
Retrieve the names of all algorithms that the smart card supports, for example "RSA," "DSA."	> Implement `CryptoSmartCard.getAlgorithms()`.
Retrieve a CryptoToken that supports the given algorithm.	> Implement `CryptoSmartCard.getCryptoToken(String)`.

Code sample: Creating a smart card object

Example: MyCryptoSmartCard.java

```
/**
* MyCryptoSmartCard.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
*/
package com.rim.samples.device.smartcard;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;
import net.rim.device.api.crypto.*;
import net.rim.device.api.ui.component.*;

/**
* This class represents a kind (or model or family) of a physical smart card.
* There should only be one instance of this class in the system at one time. The instance
* is managed by the SmartCardFactory.
*/
public class MyCryptoSmartCard extends CryptoSmartCard implements Persistable
{
    private final static byte MY_ATR [] = { (byte)0x3b, (byte)0x7d, (byte)0x11,
   (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x31,
   (byte)0x80, (byte)0x71, (byte)0x8e, (byte)0x64,
   (byte)0x86, (byte)0xd6, (byte)0x01, (byte)0x00,
   (byte)0x81, (byte)0x90, (byte)0x00 };

   private final static AnswerToReset _myATR = new AnswerToReset( MY_ATR );
```

```
private static final String LABEL = “RIM Sample”;
private static final String DISPLAY_SETTINGS = “Show driver properties/settings now”;
private static final String RSA = “RSA”;


/**
* Called on startup of the device. Register this driver with the smart card factory.
* When you do this, you are automatically registered with the user authenticator
* framework.
*/
public static void libMain( String args[] )
{
   SmartCardFactory.addSmartCard( new MyCryptoSmartCard() );
}


 /**
 * Retrieve the session handler for this smart card.
 * Implementations of this method should not bring up UI.
 */
protected SmartCardSession openSessionImpl( SmartCardReaderSession readerSession )
throws SmartCardException {
   return new MyCryptoSmartCardSession( this, readerSession );
 }

/**
* Determine if the file system should use this smart card object
* to communicate with a physical smart card.
* that has the given AnswerToReset.
* The system invokes this method to ascertain which smart card implementation it should
* use to communicate with a physical smart card found in a reader.
*/
protected boolean checkAnswerToResetImpl( AnswerToReset atr )
{
   return _myATR.equals( atr );
}


/**
* Retrieve a label associated with this smart card.
* The string should not include the words “smart card”, as the file system uses this
* this method to generate strings such as “Please insert your smart card”.
*/
    protected String getLabelImpl()
    {
       return LABEL;
    }



/**
* Retrieves this smart card’s capabilities
*/
protected SmartCardCapabilities getCapabilitiesImpl()
{
   return new SmartCardCapabilities( SmartCardCapabilities.PROTOCOL_T0 );
}
```

```
    /**
    * Determine if this smart card can display its settings.
    */
    protected boolean isDisplaySettingsAvailableImpl( Object context )
    {
       return true;
    }


    /**
    * Display this smart card's settings.
    * This method will be invoked from the smart card options screen when
    * the user selects the driver and chooses to view the settings of that driver.
    *
    * This method could be called from the event thread. The driver should not block
    * the event thread for long periods of time.
    *
    * @param context Reserved for future use.
    */
    protected void displaySettingsImpl( Object context )
    {
       Dialog.alert( DISPLAY_SETTINGS );
    }


    /** Retrieve the algorithms supported by this smart card.
    *
    * @return one of "RSA", "DSA", or "ECC"
    */
    public String[] getAlgorithms()
    {
       return new String [] { RSA };
    }


    /** Retrieve a crypto token that supports the given algorithm.
    * @param algorithm Name of the algorithm.
    *
    * @return Crypto Token supporting the named algorithm.
    *
    * @throws NoSuchAlgorithmException If the specified algorithm is invalid.
    * @throws CryptoTokenException If there is a token-related problem.
    */
    public CryptoToken getCryptoToken( String algorithm ) throws NoSuchAlgorithmException,
    CryptoTokenException
    {
       if ( algorithm.equals( RSA ) ) {
          return new MyRSACryptoToken();
       }

    throw new NoSuchAlgorithmException();
}}
```

Activate libMain() on startup

Task	Steps
Create a `libMain()` method.	> In your extension of the `CryptoSmartCard` class, implement the `libMain()` method.
Run the `libMain()` method when the application starts.	1. Open the BlackBerry® Integrated Development Environment. 2. Create a new project for the smart card driver. 3. In the **Workspace** window, right-click the new project. 4. Select **Properties**. 5. On the **Application** tab, in the **Project type** field, type **Library**. 6. Make sure that the **Auto-run on startup** check box is selected. 7. In the **Startup Tier** field, select the **7(Last; 3rd party apps only)** option. 8. Click **OK**.

Create a session for the smart card driver

Task	Steps
Create a smart card session.	> Extend the abstract `CryptoSmartCardSession` class.
Close a smart card session.	> Implement `SmartCardSession.closeImpl()`.
Retrieve the maximum number of login attempts.	> Implement `SmartCardSession.getMaxLoginAttemptsImpl()`.
Retrieve the smart card ID.	> Implement `SmartCardSession.getSmartCardIDImpl()`.
Retrieve the remaining number of login attempts.	> Implement `SmartCardSession.getRemainingLoginAttemptsImpl()`.
Attempt to log the BlackBerry® device user into the smart card session using a given password string.	> Implement `SmartCardSession.loginImpl(String)`.
Retrieve the key store data that is associated with the keys.	> Implement `CryptoSmartCard.getKeyStoreDataArrayImpl()`.
Retrieve random data from the internal random number generator of the smart card.	> Implement `CryptoSmartCard.getRandomBytesImpl(int maxNumBytes)`.

Code sample: Creating a smart card session

Example: MyCryptoSmartCardSession.java

```
/**
* MyCryptoSmartCardSession.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
```

```
*/

package com.rim.samples.device.smartcard;

import net.rim.device.api.crypto.*;
import net.rim.device.api.crypto.certificate.*;
import net.rim.device.api.crypto.certificate.x509.*;
import net.rim.device.api.crypto.keystore.*;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;

/**
* This class represents a communication session with a physical smart card.
*
* Over this session, Application Protocol Data Units may be exchanged with the smart card
* to provide the desired functionality.
* Do not hold open sessions when not using them; they should be short-lived.
* As a security precaution, only one open session is allowed to exist per SmartCardReader;
* subsequent openSession() requests will block until the current session is closed.
*/
public class MyCryptoSmartCardSession extends CryptoSmartCardSession
{
   public static final byte ID_PKI = (byte)0x00;
   public static final byte SIGNING_PKI = (byte)0x01;
   public static final byte ENCRYPTION_PKI = (byte)0x02;
   private static final String WAITING_MSG = “Please Wait”;
   private static final String ID_STRING = “John H. Smith”;
   private static final String ID_CERT = “ID Certificate”;
   private static final String SIGNING_CERT = “Signing Certificate”;
   private static final String ENCRYPTION_CERT = “Encryption Certificate”;


   /**
   * Construct a new MyCryptoSmartCardSession object.
   *
   * @param smartCard Smart card associated with this session
   * @param readerSession Reader session commands sent to this smart card.
   */
   protected MyCryptoSmartCardSession( SmartCard smartCard, SmartCardReaderSession
   readerSession )
   {
      super( smartCard, readerSession );
   }


   /**
   * Close this smart card session.
   *
   * Implementations should not close the underlying SmartCardReaderSession. Use this
   * method for cleaning up the session prior to its closure.
   */
   protected void closeImpl()
   {
      // Do any session cleanup needed here.
   }

   /**
   * Retrieve the maximum number of allowed login attempts.
```

```
* The method returns Integer.MAX_VALUE if an infinite number of attempts are allowed.
*/
protected int getMaxLoginAttemptsImpl() throws SmartCardException
{
   return 5;
}


/**
* Retrieve the remaining number of login attempts allowed (before the smart card will
* lock, or Integer.MAX_VALUE if the smart card will not lock.)
*/
protected int getRemainingLoginAttemptsImpl() throws SmartCardException
{
   return 4;
}


/**
* Log into the smart card with the given password.
* This method should not bring up the UI.
*/
protected boolean loginImpl( String password ) throws SmartCardException
{
   // Create a CommandAPDU which your smart card will understand
   CommandAPDU command = new CommandAPDU( (byte)0x00, (byte)0x20, (byte)0x00,
   (byte)0x00 );
   command.setLcData( password.getBytes() );
   ResponseAPDU response = new ResponseAPDU();
   sendAPDU( command, response );

   // Check for response codes specific to your smart card
   if ( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
      return true;
   }
   else if ( response.checkStatusWords( (byte)0x64, (byte)0xF8 ) ) {
      throw new SmartCardLockedException();
   }
   else {
      // Authentication failed
   return false;
   }
}

/**
* Retrieve an ID for this session's associated smart card.
*/
protected SmartCardID getSmartCardIDImpl() throws SmartCardException
{
   // Retrieve a unique ID from the card
   byte [] uniqueCardData = new byte[] { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
   0x08, 0x09, 0x0a, 0x0b };

   // Convert byte array to a long
   SHA1Digest digest = new SHA1Digest();
   digest.update( uniqueCardData );
   byte [] bytesToUse = Arrays.copy( digest.getDigest(), 0, 8 );
   long idLong = CryptoByteArrayArithmetic.valueOf( bytesToUse );
```

```
    // Using friendly display name
    return new SmartCardID( idLong , ID_STRING, getSmartCard() );
}


/**
* Retrieve random data from the smart card's internal Random Number Generator.
*/
protected byte [] getRandomBytesImpl( int maxBytes ) throws SmartCardException
{
    // Create a CommandAPDU that your smart card will understand
    CommandAPDU command = new CommandAPDU( (byte)0x00, (byte)0x4C, (byte)0x00,
    (byte)0x00, maxBytes );

    ResponseAPDU response = new ResponseAPDU();
    sendAPDU( command, response );

    // Check for response codes specific to your smart card
    if( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
        // The appropriate response code containing the random data
        return response.getData();
    }

    return null;
}


/**
* Retrieve certificates from the card.
*
* @return An array of certificates which are present on the card.
*/
protected CryptoSmartCardKeyStoreData[] getKeyStoreDataArrayImpl() throws
SmartCardException, CryptoTokenException
{
    try {
        // Show a progress dialog to the user as this operation may take a long time.
        displayProgressDialog( WAITING_MSG, 4 );
        // The certificates need to be associated with a particular card.
        SmartCardID smartCardID = getSmartCardID();

        RSACryptoToken token = new MyRSACryptoToken();
        RSACryptoSystem cryptoSystem = new RSACryptoSystem( token, 1024 );
        RSAPrivateKey privateKey;
        CryptoSmartCardKeyStoreData[] keyStoreDataArray = new
        CryptoSmartCardKeyStoreData[ 3 ];

        // This encoding would be extracted from the card using a series of APDU
        commands.
        Certificate certificate = null;

        // Extract the certificate encoding from the card.
        byte [] certificateEncoding = new byte[0];

            try {
            certificate = new X509Certificate( certificateEncoding ); }
            catch( CertificateParsingException e ) { // invalid X509 certificate }
```

```
         }

      stepProgressDialog( 1 );

      privateKey = new RSAPrivateKey( cryptoSystem, new MyCryptoTokenData(
      smartCardID, ID_PKI ) );
      keyStoreDataArray[ 0 ] = new CryptoSmartCardKeyStoreData( null, ID_CERT,
      privateKey, null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );
      stepProgressDialog( 1 );

      privateKey = new RSAPrivateKey( cryptoSystem, new MyCryptoTokenData(
      smartCardID, SIGNING_PKI ) );
      keyStoreDataArray[ 1 ] = new CryptoSmartCardKeyStoreData( null, SIGNING_CERT,
      privateKey, null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );

      stepProgressDialog( 1 );

      privateKey = new RSAPrivateKey( cryptoSystem, new MyCryptoTokenData(
      smartCardID, ENCRYPTION_PKI ) );
      keyStoreDataArray[ 2 ] = new CryptoSmartCardKeyStoreData( null, ENCRYPTION_CERT,
      privateKey, null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );
      stepProgressDialog( 1 );

      // Sleep so the user sees the last step of the progress dialog move to 100%
         try {
         Thread.sleep( 250 );
         } catch ( InterruptedException e ) {
         }

      dismissProgressDialog();
      return keyStoreDataArray;

      } catch ( CryptoUnsupportedOperationException e ) {
      } catch ( UnsupportedCryptoSystemException e ) {
   } catch ( CryptoTokenException e ) {
   }


   throw new SmartCardException();
}


/**
* Send some data to the smart card for signing or decryption.
*/
/*package*/ void signDecrypt( RSACryptoSystem cryptoSystem,
MyCryptoTokenData privateKeyData,byte[] input, int inputOffset,
byte[] output, int outputOffset ) throws SmartCardException
{
   // Check for nulls
   if ( cryptoSystem == null || privateKeyData == null || input == null || output ==
   null) {
      throw new IllegalArgumentException();
   }


   // Validate the input parameters
   int modulusLength = cryptoSystem.getModulusLength();
```

```
        if ( ( input.length < inputOffset + modulusLength ) || ( output.length <
        outputOffset + modulusLength ) ) {
            throw new IllegalArgumentException();
        }

        // Construct the response Application Protocol Data Unit
        ResponseAPDU response = new ResponseAPDU();


        // Construct the command and set its information
        // Create a CommandAPDU which your smart card will understand
        CommandAPDU signAPDU = new CommandAPDU( (byte)0x80, (byte)0x56, (byte)0x00,
        (byte)0x00, modulusLength );
        signAPDU.setLcData( input, inputOffset, input.length - inputOffset );

        // Send the command to the smartcard
        sendAPDU( signAPDU, response );


        // Validate the status words of the response
        // Check for response codes specific to your smart card
        if ( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
            byte [] responseData = response.getData();
            System.arraycopy( responseData, 0, output, outputOffset, responseData.length );
            }
        else {
            throw new SmartCardException( “Invalid response code, sw1=” +
            Integer.toHexString( response.getSW1() & 0xff ) + “ sw2=” + Integer.toHexString(
            response.getSW2() & 0xff ) );
            }
    }

}
```

Create a CryptoToken for private key operations

Task	Steps
Create a token class.	> In your application, create a class that extends an RSA, DSA, or ECC token class. For example: `final class MyRSACryptoToken extends RSACryptoToken implements Persistable`

Determine if the token can perform BlackBerry® device user authentication.	> Create a method that returns true if your token class prompts the BlackBerry device user for authentication information.
	`public boolean providesUserAuthentication()`
	`{`
	`return true;`
	`}`

Task	Steps

Determine if the token supports the chosen operation using the current crypto system.	> Create a method that returns a Boolean value that indicates if the token object supports the current crypto system.
	`public boolean isSupported( CryptoSystem cryptoSystem, int operation )`
	`{`
	`return ( operation == PRIVATE_KEY_OPERATION );`
	`}`

Determine if the given key and crypto system support the type of encryption method.	> Create a method that returns a Boolean value that indicates if the token object supports the chosen encryption method.
	`public boolean isSupportedDecryptRSA( RSACryptoSystem cryptoSystem, CryptoTokenPrivateKeyData privateKeyData )throws CryptoTokenException`
	`{`
	`return privateKeyData instanceof MyCryptoTokenData;`
	`}`

Activate raw decryption.	> Create a method that performs raw decryption.
	`public void decryptRSA( RSACryptoSystem cryptoSystem,`
	`CryptoTokenPrivateKeyData privateKeyData,byte[] input, int inputOffset,`
	`byte[] output, int outputOffset )throws CryptoTokenException`
	`{`
	`try {//signDecryptHelper is a private helper method. See the code sample for more information.`
	`signDecryptHelper( cryptoSystem, privateKeyData, input, inputOffset, output, outputOffset, DECRYPT_DESC, SmartCardSession.DECRYPT_OPERATION );`
	`}`
	`catch ( CryptoUnsupportedOperationException e ) {`
	`throw new CryptoTokenException( e.toString() );`
	`}`
	`}`

Activate raw signing.	> Create a method that performs raw signing.
	`public void signRSA(  RSACryptoSystem cryptoSystem, CryptoTokenPrivateKeyData privateKeyData, byte[] input, int inputOffset,byte[] output, int outputOffset )`
	`throws CryptoTokenException, CryptoUnsupportedOperationException`
	`{`
	`signDecryptHelper( cryptoSystem, privateKeyData, input, inputOffset, output, outputOffset, SIGN_DESC, SmartCardSession.SIGN_OPERATION );`
	`}`

Code sample: Creating a CryptoToken for private key RSA operations

Example: MyRSACryptoToken.java

```
/**
* MyRSACryptoToken.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
*/
package com.rim.samples.device.smartcard;

import net.rim.device.api.smartcard.*;
import net.rim.device.api.crypto.*;
import net.rim.device.api.crypto.keystore.*;
import net.rim.device.api.util.*;
import net.rim.device.api.crypto.certificate.x509.*;
import net.rim.device.api.crypto.certificate.*;
import net.rim.device.api.crypto.asn1.*;
import net.rim.device.api.compress.*;
import net.rim.device.api.i18n.*;

/**
* This class describes an implmentation of an RSA cryptographic token.
*
* The RIM Crypto API will use this object to perform a private key RSA operation.
* This object should delegate the operation to the smart card.
*/
final class MyRSACryptoToken extends RSACryptoToken implements Persistable
{
    private static final String DECRYPT_DESC = “The private key will be used to decrypt
    encrypted data.”;
    private static final String SIGN_DESC = “The private key will be used to generate a
    digital signature.”;

    /**
    * Constructs a new MyRSACryptoToken object.
    */
    MyRSACryptoToken()
    {
    }


    /**
    * Determine if this token does the user authentication for the system.
    *
    * If not the KeyStore will prompt for the key store password when the user
    * tries to access the private key.
    *
    * @return True if this token will prompt for the necessary
    * user authentication when private key access is requested.
    */
    public boolean providesUserAuthentication()
    {
        return true;
    }
```

```
/**
* Determine if this token supports the chosen operation using the provided system.
*
* @param cryptoSystem Crypto System to check against.
* @param operation Operation to check: either KEY_GENERATION,
* PUBLIC_KEY_OPERATION, PRIVATE_KEY_OPERATION,* or some other value
* specific to the crypto system that indicates the operation to check.
*/
public boolean isSupported( CryptoSystem cryptoSystem, int operation )
{
   return ( operation == PRIVATE_KEY_OPERATION );
}


/**
* Determines if the given key and crypto system
* support RSA encryption.
*
* @return True if the token supports RSA encryption.
*
* @param cryptoSystem Crypto system to check.
* @param privateKeyData Private key data.
*
* @throws CryptoTokenException If an error occurs with a crypto
* token or the crypto token is invalid.
*/
public boolean isSupportedDecryptRSA( RSACryptoSystem cryptoSystem,
CryptoTokenPrivateKeyData privateKeyData )throws CryptoTokenException
{
   return privateKeyData instanceof MyCryptoTokenData;
}


/**
* Perform a raw RSA decryption.
*
* @param cryptoSystem Crypto system associated with the token.
* @param privateKeyData RSA private key.
* @param input Input data.
* @param inputOffset First byte of the input data to read.
* @param output Buffer for the output data.
* @param outputOffset Position in the output buffer to receive the first written byte.
*
* @throws CryptoTokenException Thrown if an error occurs with a crypto
* token or the crypto token is invalid.
*/
public void decryptRSA( RSACryptoSystem cryptoSystem,
CryptoTokenPrivateKeyData privateKeyData,byte[] input, int inputOffset,
   byte[] output, int outputOffset )throws CryptoTokenException
{
   try {
      signDecryptHelper( cryptoSystem, privateKeyData, input, inputOffset, output,
      outputOffset, DECRYPT_DESC, SmartCardSession.DECRYPT_OPERATION );
      }
   catch ( CryptoUnsupportedOperationException e ) {
   throw new CryptoTokenException( e.toString() );
   }
}
```

```
/**
* Perform a raw RSA signing.
*
* @param cryptoSystem Cypto system associated with the token.
* @param privateKeyData RSA private key.
* @param input Input data.
* @param inputOffset First byte of the input data to read.
* @param output The buffer for the output data.
* @param outputOffset Position in the output buffer to receive the first written byte.
*
* @throws CryptoTokenException If an error occurs with the crypto
* token or the crypto token is invalid.
* @throws CryptoUnsupportedOperationException If a call is made to
* an unsupported operation.
*/
public void signRSA(    RSACryptoSystem cryptoSystem, CryptoTokenPrivateKeyData
privateKeyData, byte[] input, int inputOffset,byte[] output, int outputOffset )
throws CryptoTokenException, CryptoUnsupportedOperationException
{
   signDecryptHelper( cryptoSystem, privateKeyData, input, inputOffset, output,
   outputOffset, SIGN_DESC, SmartCardSession.SIGN_OPERATION );
}


/**
* Help signing and decryption operations.
* This helper method assists data signing and decryption because
* the operations are very similar.
*/
private void signDecryptHelper( RSACryptoSystem cryptoSystem,
CryptoTokenPrivateKeyData privateKeyData, byte[] input, int inputOffset,
byte[] output, int outputOffset,String accessReason,int operation )
throws CryptoTokenException, CryptoUnsupportedOperationException
{
   SmartCardSession smartCardSession = null;

   try {

      if( privateKeyData instanceof MyCryptoTokenData ) {

      SmartCardID smartCardID = ((MyCryptoTokenData) privateKeyData
      ).getSmartCardID();

      smartCardSession = SmartCardFactory.getSmartCardSession( smartCardID );

      if ( smartCardSession instanceof MyCryptoSmartCardSession ) {
      MyCryptoSmartCardSession mySmartCardSession = ( MyCryptoSmartCardSession
      )smartCardSession;

      // We must provide the user authentication since we returned true from
      //providesUserAuthentication()
      // Also, the smart card PIN is required for private key access.
      mySmartCardSession.loginPrompt( accessReason, operation );
      mySmartCardSession.signDecrypt( cryptoSystem, (MyCryptoTokenData)privateKeyData,
      input, inputOffset, output, outputOffset );
```

```
            return;
            }
        }


              throw new RuntimeException();

            } catch ( SmartCardSessionClosedException e ) {
                throw new CryptoTokenCancelException( e.toString() );
                } catch ( SmartCardCancelException e ) {
                throw new CryptoTokenCancelException( e.toString() );
            } catch( SmartCardRemovedException e ) {
                throw new CryptoTokenCancelException( e.toString() );
            } catch ( SmartCardException e ) {
                throw new CryptoTokenException( e.toString() );
            } finally {
                if ( smartCardSession != null ) {
                smartCardSession.close();


            }
        }
    }
}
```

Store the private key file location

Task	Steps
Store the location of the private key on the smart card.	> In your application, implement the `CryptoTokenPrivateKeyData` interface.

Associate the implementing class object with the smart card that contains the private key.	1. Create an instance variable for storing the smart card ID.
	`private SmartCardID _id;`
	2. Create an instance variable for storing the location of the private key file on the smart card.
	`private byte _file;`
	3. Create a method that associates an object from the class that implements a `PrivateKeyData` interface with the smart card.
	`public MyCryptoTokenData( SmartCardID id, byte file )`
	`{_id = id;`
	`_file = file;`
	`}`

Retrieve the ID of the key file that contains the private key file.	> Create a method that returns the `SmartCardID` instance variable.
	`public SmartCardID getSmartCardID()`
	`{`
	`return _id;`
	`}`

Retrieve the location of the private key file on the smart card.	> Create a method that returns the private key file smart card location instance variable.
	`public byte getFile()`
	`{`
	`return _file;`
	`}`

Code sample: Storing the location of a private key file on the smart card

Example: MyCryptoTokenData.java

```
/**
* MyCryptoTokenData.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
*/

package com.rim.samples.device.smartcard;

import net.rim.device.api.crypto.*;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;

/**
* This class stores the location of the private key file on the smart card.
*
*/
final class MyCryptoTokenData implements CryptoTokenPrivateKeyData, Persistable
{
    /**
    * Smart card containing the private key.
```

```
    */
    private SmartCardID _id;


    /**
    * Location of the private key file on the smart card.
    */
    private byte _file;


    /**
    * Constructs a new MyCryptoTokenData object
    *
    * @param id ID of the smart card containing the private key file
    * @param file Location of the private key file.
    */
    public MyCryptoTokenData( SmartCardID id, byte file )
    {
       _id = id;
       _file = file;
    }


    /**
    * Retrieve the ID of the key file containing the private key file.
    *
    * @return ID of the smart card.
    */
    public SmartCardID getSmartCardID()
    {
       return _id;
    }



    /**
    * Retrieve the location of the private key file on the smart card.
    *
    * @return Location of the private key file.
    */
    public byte getFile()
    {
       return _file;
    }

}
```

6

Managing memory

Invoking a garbage collection operation
Reduce the number of objects
Managing low memory

Invoking a garbage collection operation

See the *Garbage Collection in the BlackBerry Java Development Environment* white paper and the *BlackBerry Java Development Environment Fundamentals Guide* for more information about garbage collection operations.

Reduce the number of objects

To use the BlackBerry® Integrated Development Environment to identify unnecessary objects, complete the following steps:

1. Open the BlackBerry IDE.
2. Place two breakpoints in the code surrounding an area of high object creation.
3. Run the application to the first breakpoint.
4. Open the **Objects** window and click **Snapshot**.
5. Run the application to the second breakpoint.
6. Open the **Objects** window.
7. Click **Compare to Snapshot**.
8. View multiple snapshots in the **Objects** window.
9. Determine which objects can be removed.

Managing low memory

LMM triggers

The following conditions can cause the LMM to free memory resources:

Condition	Description
Available flash memory falls below acceptable thresholds.	The free flash memory threshold depends on the amount of free RAM in the system. The free flash memory threshold varies between 400 KB and 800 KB.

Condition	Description
A low number of persistent object handles exist on a BlackBerry device.	The number of persistent object handles falls below 1000.
A low number of object handles exist on a BlackBerry device.	The number of object handles falls below 1000.

Use the LMM

Task	Steps
Register your application with the LMM.	1. In the application, implement the `LowMemoryListener` method. 2. Enable the application to register the `LowMemoryListener` with the LMM when the application starts for the first time. Register the listener only once.
Manage events that the LowMemoryListener receives.	> Implement the `freeStaleObject(int)` method of the `LowMemoryListener` interface. The implementation of `freeStaleObject()` method should return true if persistent data is released, or return false otherwise.
Manage low priority events.	The LMM seldom specifies a priority higher than low priority. > In an implementation of `freeStaleObject()`, enable the application to release transitory variables and any variables that are currently not necessary for complete functionality, such as cached data.
Manage medium priority events.	> In an implementation of `freeStaleObject()`, enable the application to remove stale data, such as very old email messages or old calendar appointments.
Manage high priority events.	> In an implementation of `freeStaleObject()`, enable the application to remove objects in the application on a Least Recently Used basis, removing all stale objects.
Free resources manually.	The BlackBerry® device system invokes the implementation of `freeStaleObject(int)` when device memory is low. To manually free resources, perform the following actions: 1. Invoke `freeStaleObject(int)` from the application. 2. Invoke `LowMemoryManager.Poll`.

Free persistent objects

Task	Steps
Remove references to the object.	1. In the application, remove references to the persistent object. 2. Delete the object from its data structure.
Inform the JVM that it can now remove the object.	> In an implementation of `freeStaleObject()`, invoke `LowMemoryManager.markAsRecoverable()`.
Commit changes to data collections.	> In an implementation of `freeStaleObject()`, invoke `PersistentObject.commit()`.

7

Creating connections

Fetching data using HTTP or TCP sockets
Datagram connections
Using port connections

Fetching data using HTTP or TCP sockets

Java™ applications for BlackBerry® devices can use standard HTTP, HTTPS, and TCP socket protocols to establish connections over the wireless network. When establishing the connection over the cellular network, an application can use one of two wireless gateways to proxy the connection to the Internet or the corporate intranet. You can design your application to rely on the default gateway that is available to the BlackBerry device user, or you can customize your code to explicitly select a preferred gateway.

Explicitly selecting a gateway

Set up your application to use the preferred gateway for a connection and the default gateway only when the preferred gateway is not available.

Using the BlackBerry Enterprise Server as an intranet gateway

Enterprise organizations host the BlackBerry® Enterprise Server behind their corporate firewall to let BlackBerry devices access the corporate intranet. The BlackBerry Mobile Data System™ component of the BlackBerry Enterprise Server includes the BlackBerry MDS™ Services, which provides an HTTP and TCP/IP proxy service to let third-party Java™ applications use it as a secure gateway for managing HTTP and TCP/IP connections to the intranet. When you use the BlackBerry Enterprise Server as an intranet gateway, all data traffic between your application and the BlackBerry Enterprise Server is automatically encrypted using AES or Triple DES encryption. Because the BlackBerry Enterprise Server resides behind the corporate firewall and provides inherent data encryption, applications can communicate with application servers and web servers that reside on the corporate intranet.

If your application connects to the Internet rather than to the corporate intranet, you might be able to use the BlackBerry Enterprise Server that belongs to the organization as a gateway as well. In this case, network requests travel behind the corporate firewall to the BlackBerry Enterprise Server, which makes the network request to the Internet through the corporate firewall. However, enterprise customers can set an IT policy to enforce that the BlackBerry Enterprise Server is the gateway for all wireless network traffic, including traffic destined for the Internet.

If your application connects to the Internet and you are creating your application for non-enterprise organizations, you can also use either the BlackBerry Internet Service or the Internet gateway of the wireless server provider to manage connections.

Using the wireless service providers Internet gateway

Java™ applications for BlackBerry® devices can connect to the Internet using the Internet gateway that the wireless service provider provides. Most wireless service providers provide their own Internet gateway that offers direct TCP/IP connectivity to the Internet. Some operators also provide a WAP gateway that lets HTTP connections occur over the WAP protocol. Java applications for BlackBerry devices can use either of these gateways to establish connections to the Internet. If you write your application for BlackBerry device users who are on a specific wireless network, this approach can often yield good results. However, if you write your application for BlackBerry device users on a variety of wireless networks, testing your application against the different Internet gateways and achieving a consistent experience can be challenging. In these scenarios, you might find it useful to use the BlackBerry Internet Service, and use the wireless service provider's Internet gateway as a default connection type if the BlackBerry Internet Service is not available.

In the Technical Knowledge Center on the BlackBerry Developer Zone, see the *Managing Wireless Data Transport in the BlackBerry Solution v4.0 Part 1: Understanding TCP and HTTP transport options for Java applications for BlackBerry* white paper for more information about managing wireless connectivity and how to effectively use each of the gateways.

Use HTTP connections

Task	Steps
Before opening an HTTP connection, verify that the BlackBerry® device is in a wireless coverage area.	> Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless coverage area.

Open an HTTP connection.	1. Invoke `Connector.open()`, specifying **HTTP** as the protocol.
	2. Cast the returned object as an `HttpConnection` or a `StreamConnection` object.
	`HttpConnection conn = null;`
	`String URL = "http://www.myServer.com/myContent";`
	`conn = (HttpConnection)Connector.open(URL);`

Set the HTTP request method (GET or POST).	> Invoke `HttpConnection.setRequestMethod()`. `conn.setRequestMethod(HttpConnection.POST);`
Set header fields.	> Invoke `setRequestProperty()` on the `HttpConnection`. `conn.setRequestProperty("User-Agent", "BlackBerry/3.2.1");`
Retrieve header fields.	> Invoke `getRequestProperty()` on the `HttpConnection`. `String lang = conn.getRequestProperty("Content-Language");`

Send and receive data.	> Invoke `openInputStream()` and `openOutputStream()` on the `HTTPConnection`.
	`InputStream in = conn.openInputStream();`
	`OutputStream out = conn.openOutputStream();`

Code sample: Using an HTTP connection to retrieve data

The HTTPFetch.java code sample requires that you create resource files in the application project and define the required resource keys. See "Localizing applications" on page 235 for more information about creating resource files.

Example: HTTPFetch.java

```
/**
 * HTTPFetch.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httpfetch;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.io.*;
import com.rim.samples.docs.resource.*;

public class HTTPFetch extends UiApplication implements HTTPFetchResource
{
    // Constants.
    private static final String SAMPLE_PAGE = “http://localhost/testpage/sample.txt”;
    private static final String[] HTTP_PROTOCOL = {“http://”, “http:\\”};

    // Members.
    private MainScreen _mainScreen;
    private RichTextField _content;

    /**
    * Send and receive data over the network on a
    * separate thread from the main thread of your application.
    */
    ConnectionThread _connectionThread = new ConnectionThread();

    //statics
    private static ResourceBundle _resources = ResourceBundle.getBundle(
    HTTPFetchResource.BUNDLE_ID, HTTPFetchResource.BUNDLE_NAME);

    public static void main(String[] args) {
        HTTPFetch theApp = new HTTPFetch();
        theApp.enterEventDispatcher();
    }

    /**
    * The ConnectionThread class manages the HTTP connection.
    * Fetch operations are not queued, but if a second fetch request
    * is made while a previous request is still active,
    * the second request stalls until the previous request completes.
    */
    private class ConnectionThread extends Thread {
        private static final int TIMEOUT = 500; //ms

        private String _theUrl;

        /* The volatile keyword indicates that because the data is shared,
        * the value of each variable must always be read and written from memory,
        * instead of cached by the VM. This technique is equivalent to wrapping
```

```
* the shared data in a synchronized block, but produces less overhead.
*/
private volatile boolean _start = false;
private volatile boolean _stop = false;

/**
* Retrieve the URL. The synchronized keyword makes sure that only one
* thread at a time can call this method on a ConnectionThread object.
*/
public synchronized String getUrl() {
    return _theUrl;
}

/**
* Fetch a page. This method is invoked on the connection thread by
* fetchPage(), which is invoked in the application constructor when
* the user selects the Fetch menu item.
*/
public void fetch(String url) {
    _start = true;
    _theUrl = url;
}

/**
 *  Close the thread. Invoked when the application exits.
 */
public void stop() {
    _stop = true;
}

/**
* Open an input stream and extract data. Invoked when the thread
* is started.
*/
public void run() {
    for(;;) {
        // Thread control.
        while( !_start && !_stop) {
            // No connections are open for fetch requests,
            // but the thread has not been stopped.
            try {
                sleep(TIMEOUT);
            } catch (InterruptedException e) {
                System.err.println(e.toString());
            }
        }
        // Exit condition.
        if ( _stop ) {
            return;
        }
        /* Ensure that fetch requests are not missed
         * while received data is processed.
         */
        synchronized(this) {
            // Open the connection and extract the data.
            StreamConnection s = null;
            try {
                s = (StreamConnection)Connector.open(getUrl());
```

```
                        InputStream input = s.openInputStream();
                        // Extract data in 256 byte chunks.
                        byte[] data = new byte[256];
                        int len = 0;
                        StringBuffer raw = new StringBuffer();
                        while ( -1 != (len = input.read(data)) ) {
                            raw.append(new String(data, 0, len));
                        }
                        String text = raw.toString();

                        updateContent(text);

                        input.close();
                        s.close();
                    } catch (IOException e) {
                        System.err.println(e.toString());
                        // Display the text on the screen.
                        updateContent(e.toString());
                    }
                    // Reset the start state.
                    _start = false;
                }
            }
        }
    }

    private final class HTTPMainScreen extends MainScreen
    {
        // Close the connection thread when the user closes the application.
        public void close() {
            _connectionThread.stop();
            super.close();
        }
    }

    // Constructor.
    public HTTPFetch() {
        _mainScreen = new HTTPMainScreen();
        _mainScreen.setTitle(new LabelField(
        _resources.getString(APPLICATION_TITLE), LabelField.ELLIPSIS
        | LabelField.USE_ALL_WIDTH));
        _mainScreen.add(new SeparatorField());
        _content = new RichTextField(
        _resources.getString(HTTPDEMO_CONTENT_DEFAULT));
        _mainScreen.add(_content);

        // Start the helper thread.
        _connectionThread.start();
        pushScreen(_mainScreen);
        fetchPage(SAMPLE_PAGE);
    }

    // Retrieve web content.
    private void fetchPage(String url) {
        // Perform basic validation (set characters to lowercase and add http:// or https:/
/).
        String lcase = url.toLowerCase();
        boolean validHeader = false;
```

```
        int i = 0;
        for (i = HTTP_PROTOCOL.length - 1; i >= 0; --i) {
            if ( -1 != lcase.indexOf(HTTP_PROTOCOL[i]) ) {
                validHeader = true;
                break;
            }
        }
        if ( !validHeader ) {
             // Prepend the protocol specifier if it is missing.
            url = HTTP_PROTOCOL[0] + url;
        }

        // Create a new thread for connection operations.
        _connectionThread.fetch(url);
    }
    // Display the content.
    private void updateContent(final String text) {
        /* This technique creates several short-lived objects but avoids
        * the threading issues involved in creating a static Runnable and
        * setting the text.
        */
        UiApplication.getUiApplication().invokeLater(new Runnable() {
            public void run() {
                _content.setText(text);
            }
        });
    }
}
```

Use HTTP authentication

Task	Steps
Before opening an HTTP connection, verify that the BlackBerry® device is in a wireless coverage area.	> Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless coverage area.

Open an HTTP connection.	1. Invoke `Connector.open()`, using the HTTP location of the protected resource.
	2. Cast and store the returned object as a `StreamConnection`.
	`StreamConnection s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt");`
	3. Cast and store the `StreamConnection` object as an `HTTPConnection` object.
	`HttpConnection httpConn = (HttpConnection)s;`

Determine the status of the HTTP connection.	> Invoke `HttpConnection.getResponseCode()`. `int status = httpConn.getResponseCode();`

Task	Steps

Retrieve login information from a BlackBerry device user.	1. Create code that manages an unauthorized `HTTP connection` attempt.
	`int status = httpConn.getResponseCode();`
	`switch (status)`
	`case (HttpConnection.HTTP_UNAUTHORIZED):`
	2. Create a `run()` method and within it implement a `dialog` object to ask the BlackBerry device user for login information.
	`UiApplication.getUiApplication().invokeAndWait(new Runnable())`
	`{`
	`public void run()`
	`{`
	`dialogResponse = Dialog.ask;`
	`(Dialog.D_YES_NO,"Unauthorized Access:\n Do you wish to log in?");`
	`}`
	`}`

Process the response of the BlackBerry device user.	1. Create code that manages a Yes dialog response.
	2. Retrieve the login information and close the current connection.
	`if (dialogResponse == Dialog.YES)`
	`{String login = "username:password";`
	`//Close the connection.`
	`s.close();`
	3. Encode the login information.
	`byte[] encoded = Base64OutputStream.encode(login.getBytes(), 0, login.length(), false, false);`

Use the BlackBerry device user login information to access the protected resource.	> Open a new `HTTPConnection` and add the authorization header by invoking `HTTPConnection.setRequestProperty()` using the encoded login information.
	`s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt ");`
	`httpConn = (HttpConnection)s;`
	`httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));`

Code sample: Using HTTP authentication to connect to a protected Internet resource

Example: Using HTTP authentication to connect to a protected Internet resource

```
HttpConnection httpConn = null;
StreamConnection s = null;
boolean keepGoing = true;
int dialogResponse;

try
{
s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt");
httpConn = (HttpConnection)s;

while(keepGoing)
```

```
{
   int status = httpConn.getResponseCode();
   switch (status)
   {
      case (HttpConnection.HTTP_OK):
         //Connection is 200 OK.
         //Download and process data.
         keepGoing = false;
         break;


      case (HttpConnection.HTTP_UNAUTHORIZED):
         //Connection is 401 UnAuthorized.
         //A login and password is required.

         //Retrieve the login information from somewhere.
         //You could prompt the user for this information or
         //retrieve this from elsewhere if it is saved within
         //your application.
         //Login information is hard coded here for brevity, but
         //we ask the user if they want to log-in.

         UiApplication.getUiApplication().invokeAndWait(new Runnable()


         {
         public void run()
         {
         dialogResponse = Dialog.ask
         (Dialog.D_YES_NO,"Unauthorized Access:\n Do you wish to log in?");
         }
         });


            if (dialogResponse == Dialog.YES)
            {
            String login = "username:password";

            //Close the connection.
            s.close();

            //Encode the login information in Base64 format.
            byte[] encoded = Base64OutputStream.encode(login.getBytes(), 0,
            login.length(), false, false);

            //Open a new connection.
            s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt
            ");
            httpConn = (HttpConnection)s;

            //Add the authorized header.
            httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));
            }



            else
            {
            //Handle failed connection.
```

```
                keepGoing = false;
                }

                break;

                default:
                //The connection failed for some other reason.
                //Handle failed connection.
                keepGoing = false;
                break;
                }


            }
            //Close the connection.
            s.close();
        }
        catch (IOException e)
        {
        //Handle the exception.
}
```

Use HTTPS connections

Task	Steps
Before opening an HTTPS connection, verify that the BlackBerry® device is within a wireless coverage area.	> Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless coverage area.
Open an HTTPS connection.	1. Invoke `Connector.open()`, specifying **HTTPS** as the protocol. 2. Cast the returned object as an `HttpsConnection` object. `HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/");`
Specify the connection mode.	> To open an HTTPS connection in end-to-end mode, add one of the following parameters to the connection string that passes to `Connector.open()`: • Specify that an end-to-end HTTPS connection must be used from the BlackBerry device to the target server: `EndToEndRequired`. • Specify that an end-to-end HTTPS connection should be used from the BlackBerry device to the target server. If the BlackBerry device does not support end-to-end TLS, and the BlackBerry device user permits proxy TLS connections, then a proxy connection is used: `EndToEndDesired`. `HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/;EndToEndDesired");`

Use socket connections

Although you can implement HTTP over a socket connection, you should use an HTTP connection for the following reasons:

- Socket connections do not support the BlackBerry® Mobile Data System™ features, such as push.

- Applications that use socket connections typically require significantly more bandwidth than applications that use HTTP connections.

Task	Steps
Before opening a socket connection, verify that the BlackBerry device is in a wireless coverage area.	> Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless coverage area.

Open a Socket connection.	> Invoke `Connector.open()`, specifying **socket** as the protocol. Applications must input their local machine IP explicitly because localhost is not supported.
	`private static String URL = "socket://<local machine IP>:4444";`
	`StreamConnection conn = null;`
	`conn = (StreamConnection)Connector.open(URL);`

Send and receive data.	> Invoke `openInputStream()` and `openOutputStream()`.
	`OutputStreamWriter _out = new OutputStreamWriter(conn.openOutputStream());`
	`String data = "This is a test";`
	`int length = data.length();`
	`_out.write(data, 0, length);`
	`InputStreamReader _in = new InputStreamReader(conn.openInputStream());`
	`char[] input = new char[length];`
	`for ( int i = 0; i < length; ++i ) {`
	`input[i] = (char)_in.read();`
	`};`

Close the socket connection.	> Invoke `close()` on the input and output streams and the socket connection.
	`_in.close();`
	`_out.close();`
	`conn.close();`
	Each of the `close()` methods throws an IOException. Make sure that the application implements exception handling.

Datagram connections

Datagrams are independent packets of data that applications send over networks. A `Datagram` object is a wrapper for the array of bytes that is the payload of the datagram. Use a datagram connection to send and receive datagrams.

Use datagram connections

To use a datagram connection, you must have your own infrastructure to connect to the wireless network, including an APN for GPRS networks. Using UDP connections requires that you work closely with service providers. Verify that your service provider supports UDP connections.

Task	Steps
Before opening a datagram connection, verify that the BlackBerry® device is in a wireless coverage area.	> Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless coverage area. Even though the `CoverageInfo` class and the `CoverageStatusListener` interface can determine if the BlackBerry device that your application is on is in a wireless coverage area, they cannot guarantee that a subsequent network connection will be successful.
Open a datagram connection.	1. Invoke `Connector.open()`, specifying **udp** as the protocol. 2. Cast the returned object as a `DatagramConnection` object. `(DatagramConnection)Connector.open("udp://host:dest_port[;src_port]/apn");` where: • **host** is the host address in dotted ASCII-decimal format. • **dest-port** is the destination port at the host address (optional for receiving messages). • **src-port** is the local source port (optional). • **apn** is the network APN in string format.
Receive datagrams from all ports at the specified host.	> Omit the destination port in the connection string.
Open a datagram connection on a non-GPRS network.	> Specify the source port number, including the trailing slash mark. For example, the address for a CDMA network connection would be udp://121.0.0.0:2332;6343/. You can send and receive datagrams on the same port number.
Create a datagram.	> Invoke `DatagramConnection.newDatagram()`. `Datagram outDatagram = conn.newDatagram(buf, buf.length);`

Add data to a datagram.	> Invoke `Datagram.setData()`.
	`byte[] buf = new byte[256];`
	`outDatagram.setData(buf, buf.length);`

Send data on the datagram connection.	> Invoke `send()` on the datagram connection. `conn.send(outDatagram);` If an application attempts to send a datagram on a datagram connection and the recipient is not listening on the specified source port number, an `IOException` is thrown. Make sure that the application implements exception handling.

Receive data on the datgram connection.	> Invoke `receive()` on the datagram connection. Since the `receive()` method blocks other operations until it receives a data packet, use a timer to retransmit the request or close the connection if a reply does not arrive.
	`byte[] buf = new byte[256];`
	`Datagram inDatagram = conn.newDatagram(buf, buf.length);`
	`conn.receive(inDatagram);`

Extract data from a datagram.	> Invoke `getData()`. If you know the type of data that you are receiving, convert the data to the appropriate format. `String received = new String(inDatagram.getData());`
Close the datagram connection.	> Invoke `close()` on the input and output streams and on the datagram connection object.

Using port connections

Using a serial or USB connection, BlackBerry device ® applications can communicate with computer applications when they are connected to a computer using a serial or USB port. This type of connection also lets applications communicate with a peripheral device that plugs into the serial or USB port.

Use USB or serial port connections

Task	Steps
Open a USB or serial port connection.	> Invoke `Connector.open()`, specifying **comm** as the protocol and COM1 or USB as the port type. `private StreamConnection _conn = (StreamConnection)Connector.open( "comm:COM1;baudrate=9600;bitsperchar=8;parity=none;stopbits=1");`

Send data on the USB or serial port connection.	1. Invoke `openDataOutputStream()` or `openOutputStream()`.
	`DataOutputStream _dout = _conn.openDataOutputStream();`
	2. Use the write methods on the output stream to write data.
	`private String data = "This is a test";`
	`_dout.writeChars(data);`

Receive data on the USB or serial port connection.	Use a non-main event thread to read data from the input stream.
	1. Invoke `openInputStream()` or `openDataInputStream()`.
	`DataInputStream _din = _conn.openInputStream();`
	2. Use the read methods on the input stream to read data.
	`String contents = _din.readUTF();`

Close the USB or serial port connection.	1. Invoke `close()` on the input and output streams and on the port connection object.
	2. The `close()` method can throw `IOExceptions`. Make sure that the application implements exception handling.
	`_din.close();`
	`_dout.close();`
	`conn.close();`

Use Bluetooth serial port connections

You can use the Bluetooth® API (`net.rim.device.api.bluetooth`) to let your application access the Bluetooth Serial Port Profile and initiate a server or client Bluetooth serial port connection to a computer or other Bluetooth enabled device.

Task	Steps

Open a Bluetooth connection.	> Invoke `Connector.open()`, providing the serial port information that `BluetoothSerialPort.getSerialPortInfo()` returns as a parameter.
	`BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo();`
	`StreamConnection _bluetoothConnection = (StreamConnection)Connector.open( info[0].toString(), Connector.READ_WRITE );`

Task	Steps

Send data on the Bluetooth connection.	1. Invoke `openDataOutputStream()` or `openOutputStream()`.
	`DataOutputStream _dout = _bluetoothConnection.openDataOutputStream();`
	2. Use the write methods on the output stream to write data.
	`private static final int JUST_OPEN = 4;`
	`_dout.writeInt(JUST_OPEN);`

Receive data on the Bluetooth connection.	1. Create a non-main event thread to read data from the input stream.
	2. Invoke `openInputStream()` or `openDataInputStream()`.
	`DataInputStream _din = _bluetoothConnection.openDataInputStream();`
	3. Use the read methods on the input stream to read the data.
	`String contents = _din.readUTF();`

Close the Bluetooth connection.	1. Invoke `close()` on the input and output streams and on the Bluetooth serial port connection object.
	2. The `close()` method can throw `IOExceptions`. Make sure that the application implements exception handling.
	`if (_bluetoothConnection != null) {`
	`try {`
	`_bluetoothConnection.close();`
	`} catch(IOException ioe) {`
	`}`
	`}`
	`if (_din != null) {`
	`try {`
	`_din.close();`
	`} catch(IOException ioe) {`
	`}`
	`}`
	`if (_dout != null) {`
	`try {`
	`_dout.close();`
	`} catch(IOException ioe) {`
	`}`
	`}`
	`_bluetoothConnection = null;`
	`_din = null;`
	`_dout = null;`

Code sample: Listening for data on the serial port and rendering the data when it arrives

Example: BluetoothSerialPortDemo.java

```
/**
 * BluetoothSerialPortDemo.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

 /* The client side of a simple serial port demonstration application.
  * This application listens for text on the serial port and
  * renders the data when it arrives.
```

```
  */

package com.rim.samples.docs.bluetoothserialportdemo;

import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.bluetooth.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.resource.*;

public class BluetoothSerialPortDemo extends UiApplication implements
BluetoothSerialPortDemoResResource
{
    //statics ------------------------------------------------------------------
    private static ResourceBundle _resources;

    private static final int INSERT = 1;
    private static final int REMOVE = 2;
    private static final int CHANGE = 3;
    private static final int JUST_OPEN = 4;
    private static final int CONTENTS = 5;
    private static final int NO_CONTENTS = 6;




    static {
       _resources = ResourceBundle.getBundle(BluetoothSerialPortDemoResResource.BUNDLE_ID,
BluetoothSerialPortDemoResResource.BUNDLE_NAME);
    }



    private EditField _infoField;
    private StreamConnection _bluetoothConnection;
    private DataInputStream _din;
    private DataOutputStream _dout;



    private final class BluetoothDemoScreen extends MainScreen
    {
        protected void makeMenu(Menu menu, int instance)
        {
            if (_infoField.getTextLength() > 0) {
                menu.add(new MenuItem(_resources, MENUITEM_COPY, 100000, 10) {
                                public void run() {
                                    Clipboard.getClipboard().put(_infoField.getText());
                                }
                            });
            }
            super.makeMenu(menu, instance);
        }
```

```
        public void close()
        {
            closePort();
            super.close();
        }
    }



    public static void main(String[] args)
    {
        new BluetoothSerialPortDemo().enterEventDispatcher();
    }



    //constructor ---------------------------------------------------------------
    public BluetoothSerialPortDemo()
    {
        MainScreen mainScreen = new BluetoothDemoScreen();
        mainScreen.setTitle(new LabelField(_resources.getString(TITLE),
LabelField.USE_ALL_WIDTH));

        _infoField = new EditField(Field.READONLY);
        mainScreen.add(_infoField);

        pushScreen(mainScreen);

        invokeLater(new Runnable() {
            public void run() {
                openPort();
            }
        });
    }



    // Close the serial port.
    private void closePort() {
        if (_bluetoothConnection != null) {
            try {
                _bluetoothConnection.close();
            } catch(IOException ioe) {
            }
        }
        if (_din != null) {
            try {
                _din.close();
            } catch(IOException ioe) {
            }
        }
        if (_dout != null) {
            try {
                _dout.close();
            } catch(IOException ioe) {
```

```
            }
        }
        _bluetoothConnection = null;
        _din = null;
        _dout = null;
    }



    // Open the serial port.
    private void openPort() {
        if (_bluetoothConnection != null) {
            closePort();
        }
        new InputThread().start();
    }


   private class InputThread extends Thread {

   public void run() {
   try {
    BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo();
    if( info == null || info.length == 0 ) {
     invokeAndWait( new Runnable() {
      public void run() {
       Dialog.alert( “No bluetooth serial ports available for connection.” );
                     closePort();
                     System.exit(1);
                       }
                   });
               }


     _bluetoothConnection = (StreamConnection)Connector.open( info[0].toString(),
Connector.READ_WRITE );
     _din = _bluetoothConnection.openDataInputStream();
     _dout = _bluetoothConnection.openDataOutputStream();

     } catch(IOException e) {
     invokeAndWait( new Runnable() {
      public void run() {
       Dialog.alert(“Unable to open serial port”);
                      closePort();
                      System.exit(1);
                   }
               });
     } catch( UnsupportedOperationException e ) {
      invokeAndWait( new Runnable() {
       public void run() {
        Dialog.alert(“This handheld or simulator does not support bluetooth.”);
                       closePort();
                       System.exit(1);
                   }
               });
           }
```

```
        try {
            int type, offset, count;
            String value;
            _dout.writeInt(JUST_OPEN);
            _dout.flush();
            for (;;) {
                type = _din.readInt();
                if (type == INSERT) {
                    offset = _din.readInt();
                    value = _din.readUTF();
                    insert(value, offset);
                } else if (type == REMOVE) {
                    offset = _din.readInt();
                    count = _din.readInt();
                    remove(offset, count);
                } else if (type == JUST_OPEN) {
                    // Send contents to desktop.
                    value = _infoField.getText();
                    if (value == null || value.equals(“”)) {
                        _dout.writeInt(NO_CONTENTS);
                        _dout.flush();
                    } else {
                        _dout.writeInt(CONTENTS);
                        _dout.writeUTF(_infoField.getText());
                        _dout.flush();
                    }
                } else if (type == CONTENTS) {
                    String contents = _din.readUTF();
                    synchronized(Application.getEventLock()) {
                        _infoField.setText(contents);
                    }
                } else if (type == NO_CONTENTS) {
                } else {
                    throw new RuntimeException();
                }
            }
        } catch(IOException ioe) {
            invokeLater(new Runnable() {
                public void run() {
                    Dialog.alert(“Problems reading from or writing to serial port.”);
                    closePort();
                    System.exit(1);
                }
            });
        }
    }

}



private void insert(final String msg, final int offset) {
    invokeLater(new Runnable() {
        public void run() {
            _infoField.setCursorPosition(offset);
            _infoField.insert(msg);
        }
```

```
        });
    }



    private void remove(final int offset, final int count) {
        invokeLater(new Runnable() {
            public void run() {
                _infoField.setCursorPosition(offset+count);
                _infoField.backspace(count);
            }
        });
    }
}
```

8

Creating notifications

Types of notification events
Add a new event source
Respond to deferred events
Cancel events
Customize system notifications for immediate events

Types of notification events

The notification API (`net.rim.device.api.notification`) lets you add custom events for your application and define the type of notifications that BlackBerry® device users receive when custom events occur.

Notification event type	Description
Immediate events	With immediate events, BlackBerry devices notify the BlackBerry device user as soon as the event occurs, using a system notification, such as a flashing LED, vibration, or tune. An application cannot request a specific type of notification. In the BlackBerry device profiles list, BlackBerry device users control how they receive notification of immediate events by choosing an active profile and setting profile options.
Deferred events	With deferred events, BlackBerry devices schedule events in a queue according to their priority. When the event occurs, applications that the event affects can provide custom notifications to the BlackBerry device user, typically by displaying a UI element, such as a dialog box. The BlackBerry devices do not provide system-wide notifications for deferred events.

Add a new event source

Task	Steps
Create a unique long ID.	1. Define a long ID for each notification event. `public static final long ID_1 = 0xdc5bf2f81374095L;` 2. Open the BlackBerry® Integrated Development Environment. 3. In the BlackBerry IDE text pane, type a string. 4. Select the string. 5. Right-click the highlighted string. 6. Click **Convert "string" to Long**.

Define a source object.	1. Define an object that provides the source for the event.
	`Object event = new Object() {`
	2. Your implementation of `toString()` returns the string to display in the profiles list.
	`public String toString() {`
	`return "Notification Demo";`
	`}`
	`}`

Add the application to the BlackBerry device profiles.	1. Invoke `NotificationsManager.registerSource()`. 2. In this method, specify a unique event ID, the source object, and, for deferred events only, one of the following priority levels: • NotificationsConstants.CRITICAL • NotificationsConstants.SENSITIVE • NotificationsConstants.IMPORTANT • NotificationsConstants.DEFAULT_LEVEL • NotificationsConstants.CASUAL

Register the event source when the BlackBerry device starts

To register the event source when the BlackBerry® device starts, create a separate project that acts as an alternative entry point to the main application. When the BlackBerry device starts, this project automatically runs as a system module and passes an argument to the application, allowing the application to perform any one-time initializations.

Task	Steps
Create an initialization project.	1. In the BlackBerry Integrated Development Environment, create a project. 2. Right-click the project, and then click **Properties**. 3. On the **Application** tab, in the **Project type** drop-down list, click **Alternate CLDC Application Entry Point**. 4. In the **Alternate entry point** drop-down list, click the event source project. 5. In the **Arguments passed to** field, type **autostartup**. 6. Select the **Auto-run on startup** option. 7. Select the **System module** option. 8. Click **OK**.

Task	Steps

Perform initializations at the alternative entry point.	Make sure that the string checked in the If statement matches the value you type in the **Arguments passed to** field in the BlackBerry IDE project.
	> In your `main()` method, perform any required initializations.
	`public static void main (String[] args) {`
	`if ( args.length > 0 && args[0].equals(“autostartup”)) {`
	`//Application runs as a system module at startup.`
	`//Perform any necessary one-time automatic initialization.`
	`} else {`
	`//Application is being run by a user.`
	`}`
	`}`

Code sample:

Example: NotificationsDemo.java

```
/**
 * NotificationsDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.notifications;

import net.rim.device.api.notification.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public class NotificationsDemo extends UiApplication
{
    public static final long ID_1 = 0xdc5bf2f81374095L;
    private long _eventIdGenerator;
    private static Object er;

    public static final Object event = new Object() {
        public String toString() {
            return “Sample Notification Event #1”;
        }
    };

    public static void main(String[] args) {
        if ( args.length > 0 && args[0].equals( “autostartup” ) ) {
            NotificationsManager.registerSource(ID_1, event,
NotificationsConstants.CASUAL);
            NotificationsManager.registerConsequence(ConsequenceDemo.ID, new
ConsequenceDemo());
        } else {
            NotificationsDemo app = new NotificationsDemo();
            app.enterEventDispatcher();
        }
```

```
    }

    public NotificationsDemo() {
        MainScreen mainScreen = new NotificationsMainScreen();
        mainScreen.setTitle("Notification Demo App");
        NotificationsManager.registerNotificationsEngineListener(ID_1,
            new NotificationsEngineListenerImpl(this));
        pushScreen(mainScreen);
    }

    private MenuItem triggerItem = new MenuItem(null, 0, 100, 10) {
        public void run() {
            NotificationsManager.triggerImmediateEvent(ID_1, 0, this, null);
        }
        public String toString() {
            return "Trigger event";
        }
    };

    private MenuItem deferItem = new MenuItem(null, 0, 100, 10) {
        public void run() {
            long timeout = -1; // Ignored unless trigger is OUT_OF_HOLSTER_TRIGGER.
            int trigger = NotificationsConstants.MANUAL_TRIGGER;
            Object er = new Object();
            NotificationsManager.negotiateDeferredEvent(ID_1, ++_eventIdGenerator,
                    er, timeout, trigger, null);
                }
        public String toString() {
            return "Start deferred event";
        }
    };

    private MenuItem cancelItem = new MenuItem(null, 0, 100, 10) {
        public void run() {
            int trigger = NotificationsConstants.MANUAL_TRIGGER;
            NotificationsManager.cancelDeferredEvent(ID_1, _eventIdGenerator, er,
                trigger, null);
            }
        public String toString() {
            return "Cancel deferred event";
        }
    };

    private final class NotificationsMainScreen extends MainScreen
    {
        protected void makeMenu( Menu menu, int instance ) {
            menu.add(triggerItem);
            menu.add(deferItem);
            menu.add(cancelItem);
            super.makeMenu(menu, instance);
        }
    }

    private static class NotificationsEngineListenerImpl implements
            NotificationsEngineListener {
        private UiApplication _app;
        public NotificationsEngineListenerImpl(UiApplication app) {
            _app = app;
```

```
        }

        public void deferredEventWasSuperseded(long sourceID, long eventID,
                Object eventReference, Object context) {
            final long _eventID = eventID;
            er = eventReference;
            _app.invokeLater(new Runnable() {
                public void run() {
                    NotificationsManager.cancelDeferredEvent(ID_1, _eventID, er,
                            NotificationsConstants.MANUAL_TRIGGER, null);
                }
            });
        }
        public void notificationsEngineStateChanged(int stateInt, long sourceID,
                long eventID, Object eventReference, Object context) {
            if(stateInt == NotificationsConstants.OUT_OF_HOLSTER_ENGINE_STATE) {
                // Perform some action if handheld is removed from holster.
            }
            if(stateInt == NotificationsConstants.IN_HOLSTER_ENGINE_STATE) {
                // Perform some action if handheld is inserted into holster.
            }
        }
        public void proceedWithDeferredEvent(long sourceID, long eventID,
                Object eventReference, Object context) {
            final long _eventID = eventID;
            _app.invokeLater(new Runnable() {
                public void run() {
                    String s = "This event has occurred:  " + _eventID;
                    Dialog d = new Dialog(Dialog.D_OK, s, Dialog.OK,
                            Bitmap.getPredefinedBitmap(Bitmap.INFORMATION), 0);
                    d.show();
                }
            });
        }
    }
}
```

Triggering events

Task	Steps
Trigger an immediate event.	> Invoke `triggerImmediateEvent()`. `NotificationsManager.triggerImmediateEvent(ID_1, 0, this, null);`
Trigger a deferred event.	1. Invoke `negotiateDeferredEvent()`. `NotificationsManager.negotiateDeferredEvent(ID_1, 0, this, -1, NotificationsConstants.MANUAL_TRIGGER, null);` 2. If you invoke `negotiateDeferredEvent(long, long, Object, long, int, Object)`, your application must implement the `NotificationEventListener` to receive events and respond appropriately.

Respond to deferred events

Task	Steps
Provide a custom UI notification.	> Implement the NotificationsEngineListener interface. `private static class ListenerImpl implements NotificationsEngineListener {...}`

Define behavior if an event is superseded by another event at the same or higher priority level.	> Implement `deferredEventWasSuperseded()`.
	`public void deferredEventWasSuperseded(long sourceID, long eventID, Object eventReference, Object context) {`
	`final long _eventID = eventID;`
	`er = eventReference;`
	`_app.invokeLater(new Runnable() {`
	`public void run() {`
	`NotificationsManager.cancelDeferredEvent(ID_1, _eventID, er,`
	`NotificationsConstants.MANUAL_TRIGGER, null);`
	`}`
	`});`
	`}`

Define behavior if the BlackBerry® device user inserts or removes the BlackBerry device from the holster.	> Implement `notificationsEngineStateChanged()`.
	`public void notificationsEngineStateChanged(int stateInt, long sourceID, long eventID, Object eventReference, Object context) {`
	`if(stateInt == notificationsConstants.OUT_OF_HOLSTER_ENGINE_STATE) {`
	`// Perform action if the BlackBerry device is removed from the holster.`
	`}`
	`if(stateInt == NotificationsConstants.IN_HOLSTER_ENGINE_STATE) {`
	`// Perform action if the BlackBerry device is inserted into the holster.`
	`}`
	`}`

Define the notification when the event occurs.	> Implement `proceedWithDeferredEvent()`.
	`public void proceedWithDeferredEvent(long sourceID, long eventID, Object eventReference, Object context) {`
	`final long _eventID = eventID;`
	`_app.invokeLater(new Runnable() {`
	`public void run() {`
	`String s = "This event has occurred:  " + _eventID;`
	`Dialog d = new Dialog(Dialog.D_OK, s, Dialog.OK,`
	`Bitmap.getPredefinedBitmap(Bitmap.INFORMATION), 0);`
	`d.show();`
	`_eventHashtable.put(_eventID, d);`
	`}`
	`});`
	`}`

Register the notifications listener with the NotificationsManager.	You can register only one `NotificationsEngineListener` for each application. > Invoke `registerNotificationsEngineListener(int, NotificationsEngineListener)`, providing as parameters the event source ID of your application and an instance of the class that implements the `NotificationsEngineListener` interface. `NotificationsManager.registerNotificationsEngineListener( ID_1, new ListenerImpl(this));`

Cancel events

Task	Steps
Cancel an immediate event.	> Invoke `cancelImmediateEvent(long, long, Object, Object)`, and then specify the source and event ID. `NotificationsManager.cancelImmediateEvent(ID_1, 0, this, null);`
Cancel a deferred event.	> Invoke `cancelDeferredEvent(long, long, Object, int, Object)`, and then specify the source and event ID. `NotificationsManager.cancelDeferredEvent(ID_1, 0, this, NotificationsConstants.MANUAL_TRIGGER, null);`
Cancel all deferred events.	If you invoke `negotiateDeferredEvent()` and do not specify a timeout, you must invoke `cancelDeferredEvent()` to cancel the event or else the event never expires. > Invoke `cancelAllDeferredEvents(long, int, Object)` to cancel all deferred events that your application starts. `NotificationsManager.cancelAllDeferredEvents(ID_1, NotificationsConstants.MANUAL_TRIGGER, null);`

Customize system notifications for immediate events

Task	Steps
Respond to notification events.	> Create a class that implements the `Consequence` and `SyncConverter` interfaces. The `SyncConverter` interface defines the functionality necessary to convert data from object to serialized format. `private static class ConsequenceImpl implements Consequence,SyncConverter {...}`
Define a unique ID.	> Define a unique ID for the consequence. `public static final long ID = 0xbd2350c0dfda2a51L;`

Define the constants.	> Declare the **DATA** and **TYPE** constants to identify data for the application. When the application invokes `convert()`, the constants identify the type of incoming data from the `SyncConverter`.
	`private static final int TYPE = 'n' << 24 | 'o' << 16 | 't' << 8 | 'd';`
	`private static final byte[] DATA = new byte[] {'m', 'y', '-', 'c', 'o', 'n', 'f', 'i', 'g', '-', 'o', 'b', 'j', 'e', 'c', 't'};`
	`private static final Configuration CONFIG = new Configuration(DATA);`

Create a tune that plays when the BlackBerry® device user receives the notification.	> Create a tune that plays as part of the consequence for event notifications.
	`private static final short BFlat = 466; // 466.16`
	`private static final short TEMPO = 125;`
	`private static final short d16 = 1 * TEMPO;`
	`private static final short dpause = 10; // 10 millisecond pause`
	`private static final short[] TUNE = new short[] {BFlat, d16, pause, BFlat};`
	`private static final int VOLUME = 80; // Percentage volume.`

Task	Steps

Define a notification.	> Implement `startNotification()`.
	`public void startNotification(long consequenceID, long sourceID, long eventID, Object configuration, Object context) {`
	`LED.setConfiguration(500, 250, LED.BRIGHTNESS_50);`
	`LED.setState(LED.STATE_BLINKING);`
	`Alert.startAudio(TUNE, VOLUME);`
	`Alert.startBuzzer(TUNE, VOLUME);`
	`}`

Stop a notification.	> Implement `stopNotification()`.
	`public void stopNotification(long consequenceID, long sourceID, long eventID, Object configuration, Object context) {`
	`LED.setState(LED.STATE_OFF);`
	`Alert.stopAudio();`
	`Alert.stopBuzzer();`
	`}`

Store the event notification user profile settings.	> Implement `newConfiguration()`.
	`public Object newConfiguration(long consequenceID, long sourceID, byte profileIndex, int level, Object context) {`
	`return CONFIG;`
	`}`

Activate data backup for the event notification user profile settings.	> Implement `SyncConverter.convert()`.
	`public SyncObject convert(DataBuffer data, int version, int UID) {`
	`try {`
	`int type = data.readInt();`
	`int length = data.readCompressedInt();`
	`if ( type == TYPE ) {`
	`byte[] rawdata = new byte[length];`
	`data.readFully(rawdata);`
	`return new Configuration(rawdata);`
	`}`
	`} catch (EOFException e) {`
	`System.err.println(e);`
	`}`
	`return null;`
	`}`

Activate data restore for the event notification user profile settings.	> Implement `SyncConverter.convert()`.
	`public boolean convert(SyncObject object, DataBuffer buffer, int version) {`
	`boolean retval = false;`
	`if ( object instanceof Configuration ) {`
	`Configuration c = (Configuration)object;`
	`buffer.writeInt(TYPE);`
	`buffer.writeCompressedInt(c._data.length);`
	`buffer.write(c._data);`
	`retval = true;`
	`}`
	`return retval;`
	`}`

Task	Steps

Define the notification configuration.	1. Create a class that implements `SyncObject` and `Persistable`.
	`private static final class Configuration implements SyncObject, Persistable {`
	2. In the class, make sure that the `SyncObject.getUID()` method returns 0 if data synchronization is not required.
	`public byte[] _data;`
	`public Configuration(byte[] data) {`
	`_data = data;`
	`}`
	`public int getUID() {`
	`return 0;`
	`}`
	`}`

Register a custom notification in the NotificationsManager.	> If you create a custom `Consequence` implementation, register it with the `NotificationsManager` by invoking `registerNotificationsObjects(long, Consequence)`. `NotificationsManager.registerConsequence(ConsequenceImpl.ID, new ConsequenceImpl());`

Code sample: Creating a custom notification

Example: ConsequenceDemo.java

```
/**
 * ConsequenceDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
*/

package com.rim.samples.docs.notifications;

import net.rim.device.api.synchronization.*;
import net.rim.device.api.notification.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.io.*;

public class ConsequenceDemo implements Consequence, SyncConverter {

    public static final long ID = 0xbd2350c0dfda2a51L;
    private static final int TYPE = 'n' << 24 | 'o' << 16 | 't' << 8 | 'd';
    private static final byte[] DATA = new byte[] {
            'm', 'y', '-', 'c', 'o', 'n', 'f', 'i',
            'g', '-', 'o', 'b', 'j', 'e', 'c', 't' };

    private static final Configuration CONFIG = new Configuration(DATA);

    private static final short BFlat = 466; // The actual value is 466.16.
    private static final short TEMPO = 125;
    private static final short d16 = 1 * TEMPO;
    private static final short pause = 10; // 10 millisecond pause.
    private static final short[] TUNE = new short[] {BFlat, d16, pause, BFlat};
    private static final int VOLUME = 80; // Percentage volume.
```

```
public void startNotification(long consequenceID, long sourceID, long eventID,
    Object configuration, Object context) {
    LED.setConfiguration(500, 250, LED.BRIGHTNESS_50);
    LED.setState(LED.STATE_BLINKING);

    Alert.startAudio(TUNE, VOLUME);
    Alert.startBuzzer(TUNE, VOLUME);
}

public void stopNotification(long consequenceID, long sourceID, long eventID,
    Object configuration, Object context) {
    LED.setState(LED.STATE_OFF);
    Alert.stopAudio();
    Alert.stopBuzzer();
}

public Object newConfiguration(long consequenceID, long sourceID,
    byte profileIndex, int level, Object context) {
    return CONFIG;
}

public SyncObject convert(DataBuffer data, int version, int UID) {
    try {
        int type = data.readInt();
        int length = data.readCompressedInt();
        if ( type == TYPE ) {
            byte[] rawdata = new byte[length];
            data.readFully(rawdata);
            return new Configuration(rawdata);
        }
    } catch (EOFException e) {
        System.err.println(e);
    }
    return null;
}
public boolean convert(SyncObject object, DataBuffer buffer, int version) {
    boolean retval = false;
    if ( object instanceof Configuration ) {
        Configuration c = (Configuration)object;
        buffer.writeInt(TYPE);
        buffer.writeCompressedInt(c._data.length);
        buffer.write(c._data);
        retval = true;
    }
    return retval;
}

/* Inner class to store configuration profile. */
private static final class Configuration implements SyncObject, Persistable {

    public byte[] _data;

    public Configuration(byte[] data) {
        _data = data;
    }
    public int getUID() {
        return 0;
```

```
            }
        }
}
```

9

Managing applications

Application manager
Retrieve information about applications
Register applications when the BlackBerry device starts
Communicate with other applications
Determine the services that are available to BlackBerry applications
Listen for changes to IT policies
Managing code modules
Runtime store
Share runtime objects

Application manager

The JVM on BlackBerry® devices includes an application manager that functions as the central dispatcher of operating system events for other Java™ applications.

The `net.rim.device.api.system.ApplicationManager` class lets applications interact with the application manager to perform the following actions:

- interact with processes, such as retrieving the IDs for foreground applications
- post global events to the system
- run an application immediately or at a specific time

Retrieve information about applications

Task	Steps

Retrieve information about the processes that are running.	> Invoke `ApplicationManager.getVisibleApplications()`.
	`ApplicationManager manager = ApplicationManager.getApplicationManager();`
	`ApplicationDescriptor descriptors[] = manager.getVisibleApplications();`

Retrieve descriptions of the objects for the applications that are running.	> Invoke `getName()`. `String appname1 = descriptors[0].getName();`
Retrieve a description of the current application.	> Invoke `ApplicationDescriptor.currentApplicationDescriptor()`. `ApplicationDescriptor descriptor = ApplicationDescriptor.currentApplicationDescriptor();`

Register applications when the BlackBerry device starts

To register the event source when the BlackBerry® device starts, create a separate project that acts as an alternative entry point to the main application. When the BlackBerry device starts, this project automatically runs as a system module and passes an argument to the application, allowing the application to perform any one-time initialization. You cannot pass arguments to MIDlet applications when the BlackBerry device starts.

Task	Steps
Retrieve information about the processes that are running.	1. In the BlackBerry Integrated Development Environment, create a project. 2. Right-click the project, and then click **Properties**. 3. Click the **Application** tab. 4. In the **Project type** drop-down list, click **Alternate CLDC Application Entry Point**. 5. In the **Alternate entry point** drop-down list, click the alternate entry point project. 6. In the **Arguments passed to** field, type **autostartup**. 7. Select the **Auto-run on startup** option. 8. Select the **System module** option. 9. Click **OK**.

Perform initialization at the alternative entry point.	> In your `main()` method, perform the required initialization. For example:
	`public static void main (String[] args)`
	`{if ( args.length > 0 && args[0].equals("autostartup")) {`
	`//Application runs as a system module at startup.`
	`//Perform any necessary one-time automatic initialization.`
	`} else {`
	`//Application is being run by a user.}`
	`}`

Communicate with other applications

> To post a system-level event to other applications, invoke one of the `ApplicationManager.postGlobalEvent()` methods.

Determine the services that are available to BlackBerry applications

The service book consists of service records, each of which defines a service on a BlackBerry® device. Service records define the communication protocol (WAP or IPPP), the network gateway, and the configuration information such as browser settings.

> To let your application interact with the BlackBerry Infrastructure, use the service book API (`net.rim.device.api.servicebook`).

Listen for changes to IT policies

Task	Steps
Enable an application to use IT policies.	> Implement the `GlobalEventListener` interface.
Identify changes in IT policies.	> Implement `GlobalEventListener.eventOccurred()`.

Code example: Listening for changes to IT policies

Example: ITPolicyDemo.java

```
/**
 * ITPolicyDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.itpolicy;


import net.rim.device.api.system.*;
import net.rim.device.api.itpolicy.*;

public class ITPolicyDemo extends Application implements GlobalEventListener {
    public static void main(String[] args) {
        ITPolicyDemo app = new ITPolicyDemo();
        app.enterEventDispatcher();
    }
    ITPolicyDemo() {
        this.addGlobalEventListener(this);
        boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled", true);
        System.out.println("App Enabled: " + appEnabled);
        System.exit(0);
    }
    public void eventOccurred(long guid, int data0, int data1, Object obj0, Object obj1) {
        if (guid == ITPolicy.GUID_IT_POLICY_CHANGED ) {
            String security = ITPolicy.getString("DemoSecurityLevel");
            boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled", true);
            int retries = ITPolicy.getInteger("DemoAppRetries", 10);
        }
    }
}
```

Managing code modules

A code module is a .cod file, the compiled archive of a single project in the BlackBerry® Integrated Development Environment.

To retrieve information about and manage code modules on the BlackBerry device, use the `CodeModuleManager` class in the `net.rim.device.api.system` package.

Retrieve module information

Task	Steps
Retrieve a handle for a module.	> Invoke `getModuleHandle()` and provide the name of the code module as a parameter. `int handle = CodeModuleManager.getModuleHandle("test_module");`

Retrieve specific information about a module.	> Invoke the methods of the `CodeModuleManager` class and provide the module handle as a parameter to these methods.
	`String name = CodeModuleManager.getModuleName( handle );`
	`String vendor = CodeModuleManager.getModuleVendor( handle );`
	`String description = CodeModuleManager.getModuleDescription( handle );`
	`int version = CodeModuleManager.getModuleVersion( handle );`
	`int size = CodeModuleManager.getModuleCodeSize( handle );`
	`int timestamp = CodeModuleManager.getModuleTimestamp( handle );`

Retrieve an array of handles for existing modules on a BlackBerry device

> Invoke `getModuleHandels()`.

```
int handles[] = CodeModuleManager.getModuleHandles();
String name = CodeModuleManager.getModuleName( handles[0]);
```

Create code modules

Task	Steps
Create a module without data.	> Invoke `createNewModule()` and provide the size of the module in bytes as a parameter. `int handle = CodeModuleManager.createNewModule( 3000 );`
Create a module with data.	> Invoke `createNewModule(int, byte[], int)`, providing the following parameters: • the length in bytes of the entire module • the byte array to add to the module • the length parameter to specify the number of bytes from the byte array to add to the start of the module `static int createNewModule(int, byte[], int);`
Write data into a module.	You can write data into a code module in increments, as long as you know the offset at which to add data. > Invoke `writeNewModule()` and provide a byte array of data as a parameter to this method. `Boolean success = CodeModuleManager.writeNewModule( handle, data, 0, data.length );`
Save a module to the BlackBerry® device database.	> Invoke `saveNewModule(int)`. If the module saves successfully, the method returns one of the result codes defined in the `CodeModuleManager` class. `int result = CodeModuleManager.saveNewModule(handle);`

Task	Steps

Delete a module from the BlackBerry device database.	> Invoke `deleteModuleEx(int, Boolean)` and provide the following parameters:
	• the handle of the module to delete
	• a Boolean value to specify whether to delete the module and any data it contains, or to delete the module only if it does not have data associated with it
	`int handle = CodeModuleManager.getModuleHandle("test_module");`
	`if( handle != 0 ) {`
	`Boolean success = CodeModuleManager.deleteModule( handle, true );`
	`}`
	If the module is in use, delete it when the BlackBerry device restarts.

Runtime store

BlackBerry® devices use a runtime store as a central location in which applications can share runtime objects. By default, only applications that Research In Motion (RIM) digitally signs can access data in the runtime store. Contact RIM for information about how to control access to your data.

The runtime store is not persistent. A BlackBerry device restart clears the data in the runtime store.

Share runtime objects

Task	Steps
Retrieve the runtime store.	> Invoke `RuntimeStore.getRuntimeStore()`. `RuntimeStore store = RuntimeStore.getRuntimeStore();`

Add a runtime object.	1. Invoke `RuntimeStore.put(long, String)` and provide as parameters a unique long ID and the runtime object to store.
	`RuntimeStore store = RuntimeStore.getRuntimeStore();`
	`// Create an object and a unique number to identify the object.`
	`String msg = "Some shared text";`
	`long ID = 0x60ac754bc0867248L;`
	2. Create a try-catch block to manage the `IllegalArgumentException` that `put()` throws if a runtime object with the same ID exists.
	`try {`
	`store.put( ID, msg );`
	`} catch(IllegalArgumentException e) {`
	`// Handle exception - an object with the same ID exists.`
	`}`

Task	Steps

Replace a runtime object.	1. Invoke `replace()`.
	`RuntimeStore store = RuntimeStore.getRuntimeStore();`
	`String newmsg = "Some new text";`
	2. Create a try-catch block to manage the `ControlledAccessException` that `replace()` throws if the runtime object with the specified ID does not exist.
	`try {`
	`Object obj = store.replace( 0x60ac754bc0867248L, newmsg);`
	`} catch(ControlledAccessException e) {`
	`// Handle exception - insufficient permissions.`
	`} not exist.`

Retrieve a registered runtime object.	1. Invoke `RuntimeStore.get()` and provide as a parameter the runtime object ID.
	`RuntimeStore store = RuntimeStore.getRuntimeStore();`
	2. Create a try-catch block to manage the `ControlledAccessException` that `get()` throws if the application does not have read access to the specified runtime object.
	`try {`
	`// get() returns the objectm with the specified ID if it exists; null`
	`// otherwise.`
	`Object obj = store.get(0x60ac754bc0867248L);`
	`} catch(ControlledAccessException e) {`
	`// Handle exception.`
	`}`

Retrieve an unregistered runtime object.	1. Invoke `RuntimeStore.waitFor()` to wait for registration of a runtime object to complete. If the runtime object with the specified ID does not exist, waitFor() blocks for a maximum of MAX_WAIT_MILLIS.
	`RuntimeStore store = RuntimeStore.getRuntimeStore();`
	2. Create code for handling exceptions.
	`try {`
	`Object obj = store.waitFor(0x60ac754bc0867248L);`
	`} catch(ControlledAccessException e) {`
	`// Handle exception - insufficient permissions.`
	`} catch(RuntimeException e) {`
	`// Handle exception - time out.`
	`}`

10

Using the messages application

Create new messages
Work with a message
Work with folders
Working with attachments

Create new messages

Task	Steps
Create a new blank text message.	> Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the ARG_NEW_SMS parameter. `Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_SMS));`

Create a new populated text message.	1. Create and populate a new `TextMessage` object.
	`MessageConnection mc = (MessageConnection)Connector.open( "sms://" );`
	`TextMessage m = (TextMessage)mc.newMessage( MessageConnection.TEXT_MESSAGE );`
	`m.setAddress( "sms://5558888" );`
	`m.setPayloadText( "An SMS Message for you" );`
	2. Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the new TextMessage object.
	`Invoke.invokeApplication( Invoke.APP_TYPE_MESSAGES, new MessageArguments( m ) );`

Create a new text message with multimedia.	> Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the ARG_NEW_MMS parameter. `Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_MMS));`
Create a new blank email message.	> Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the ARG_NEW parameter. `Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW));`

Task	Steps

Create a new populated email message.	1. Create and populate a new email `message` object.
	`net.rim.blackberry.api.mail.Message m = new net.rim.blackberry.api.mail.Message();`
	`Address a = new Address("mLi@rim.com", "Ming Li");`
	`Address[] addresses = {a};`
	`m.addRecipients(net.rim.blackberry.api.mail.Message.RecipientType.TO, addresses);`
	`m.setContent("A message for you...");`
	`m.setSubject("Email for you");`
	2. Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the new email `Message` object.
	`Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m));`

Create a new blank PIN message.	> Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the ARG_NEW_PIN parameter. `Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_PIN));`

Create a new populated PIN message.	1. Create and populate a new PIN message.
	`net.rim.blackberry.api.mail.Message m = new net.rim.blackberry.api.mail.Message();`
	`PINAddress pa = new PINAddress("ABCDEF99", "Mark Chapters");`
	`Address[] addresses = {pa};`
	`m.addRecipients( net.rim.blackberry.api.mail.Message.RecipientType.TO, addresses );`
	`m.setContent("A message for you...");`
	`m.setSubject("PIN message for you");`
	2. Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses the new PIN message.
	`Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m));`

Work with a message

Task	Steps
Receive a message notification.	1. Implement the `FolderListener` and `StoreListener` interfaces. `public class MailTest implements FolderListener, StoreListener { ... }` 2. Create code to manage a `ControlledAccessException`.

Add a listener to the message store.	1. Retrieve the `Store` object.
	2. Add a `StoreListener` instance to it.
	3. Create a try-catch block to manage a `NoSuchServiceException`.
	`try {`
	`Store store = Session.waitForDefaultSession().getStore();`
	`} catch (NoSuchServiceException e) {`
	`System.out.println(e.toString());`
	`}`
	`store.addStoreListener(this);`

Add a listener to the message store for batch updates.	> Implement `StoreListener.batchOperation()`.
	`void batchOperation(StoreEvent e) {`
	`// Perform action when messages added or removed in batch operation.`
	`}`

Add a listener to a folder.	1. Retrieve the `Folder` object for which you want to receive new message notifications.
	`Folder[] folders = store.list(Folder.INBOX);`
	`Folder inbox = folders[0];`
	2. Add the `FolderListener` instance to the folder.
	`inbox.addFolderListener(this);`
	3. Implement `FolderListener.messagesAdded()` and `FolderListener.messagesRemoved()`.
	`void messagesAdded(FolderEvent e) {`
	`// Perform processing on added messages.`
	`}`
	`void messagesRemoved(FolderEvent e) {`
	`// Perform processing on removed messages.`
	`}`

Get more of a message.	By default, the first section of a message (typically about 2 KB) is sent to the BlackBerry® device.
	1. To determine whether more data is available on the server, invoke `hasMore()` on a body part .
	2. To determine if the BlackBerry device user made a request for more data, invoke `moreRequestSent()`.
	3. To request more of a message, invoke `more()`. The second parameter of `more()` is a Boolean value that specifies whether to retrieve only the next section of the body part (false) or all remaining sections of the body part (true).
	`if (( bp.hasMore() ) && (! bp.moreRequestSent()) {`
	`Transport.more(bp, true);`
	`}`

Open a message

1. Retrieve the message store and the folder that contains the message.

```
Store store = Session.waitForDefaultSession.getStore();

Folder folder = Store.getFolder("SampleFolder");
```

2. Retrieve the message objects from the folder. Iterate through the array and retrieve information, such as the sender and subject, to display to the BlackBerry® device user.

```
Message[] msgs = folder.getMessages();
```

3. When a BlackBerry device user selects a message from the list, invoke methods on the `Message` object to retrieve the appropriate fields and body contents to display to the BlackBerry device user.

```
Message msg = msgs[0]; // Retrieve the first message.

Address[] recipients = msg.getRecipients(Message.RecipientType.TO)

Date sent = msg.getSentDate();

Address from = msg.getFrom();

String subject = msg.getSubject();

Object o = msg.getContent();

// Verify that the message is not multipart.

if ( o instanceof String ) {
String body = (String)o;} //...
```

4. Invoke `getBodyText()` on a message to retrieve the plain text contents as a `String`. If the message does not contain plain text, the method returns null.

Send a message

Task	Steps

Create a message.	1. Create a `Message` object.
	2. Specify a folder in which to save a copy of the sent message.
	`Store store = Session.getDefaultInstance().getStore();`
	`Folder[] folders = store.list(Folder.SENT);`
	`Folder sentfolder = folders[0];`
	`Message msg = new Message(sentfolder);`

Specify the recipients.	1. Create an array of `Address` objects.
	`Address[] toList = new Address[1];`
	2. Add each address to the array.
	3. Create code to catch an `AddressException` that the `Address` array throws if an address is invalid.
	`try {`
	`toList[0]= new Address("aisha.wahl@blackberry.com", "Aisha Wahl");`
	`} catch(AddressException e) {`
	`System.out.println(e.toString());`
	`}`

Task	Steps
Add the recipients.	1. Invoke `Message.addRecipients()` and provide the type of recipient (TO, CC, or BCC) and the array of addresses to add as parameters to the method. If the message has multiple types of recipients, invoke `addRecipients()` once for each recipient type. `msg.addRecipients(Message.RecipientType.TO, toList);`

Specify the name and email address of the sender.	> Invoke `setFrom(Address)`.
	`Address from = new Address("scott.mcpherson@blackberry.com", "Scott McPherson");`
	`msg.setFrom(from);`

Add a subject line.	> Invoke `setSubject(String)`. `msg.setSubject("Test Message");`

Specify the message contents.	> Invoke `setContent(String)`. Typically, the application retrieves content from text that a BlackBerry® device user types in a field.
	`try {`
	`msg.setContent("This is a test message.");`
	`} catch(MessagingException e) {`
	`System.out.println(e.getMessage());`
	`}`

Send the message.	> Invoke `Transport.send(Message)`. The `Transport` object represents the messaging transport protocol.
	`try {`
	`Transport.send(msg);`
	`} catch(MessagingException e) {`
	`System.out.println(e.getMessage());`
	`}`

Reply to a message

Task	Steps

Reply to a message.	> Invoke `Message.reply(Boolean)`. As a parameter to this method, specify true to reply to all message recipients or false to reply to only the sender.
	`Store store = Session.waitForDefaultSession().getStore();`
	`Folder[] folders = store.list(INBOX);`
	`Folder inbox = folders[0];`
	`Message[] messages = folder.getMessages();`
	`if( messages.length > 0 ) {`
	`Message msg = messages[0];`
	`}`
	`Message reply = msg.reply(true);`
	`Transport.send(reply);`

Forward a message

Task	Steps
Create a message object.	> Invoke `forward()` on a `Message` object. The subject line of a forwarded message is set automatically to FW:<*original_subject*>. `Message fwdmsg = msg.forward();`

Add the recipients.	1. Create an array of addresses.
	`Address toList[] = new Address[1];`
	2. Invoke `addRecipients(int, Address[])`.
	`toList[0]= new Address("aisha.wahl@blackberry.com", "Katie Laird");`
	`fwdmsg.addRecipients(Message.RecipientType.TO, toList);`

Specify that the message content appears before the original message.	> Invoke `setContent(String)`.
	`try {`
	`fwdmsg.setContent("This is a forwarded message.");`
	`} catch(MessagingException e) {`
	`System.out.println(e.getMessage());`
	`}`

Send the message.	> Invoke `send(Message)`.
	`try {`
	`Transport.send(fwdmsg);`
	`} catch(MessagingException e) {`
	`System.out.println(e.getMessage());`
	`}`

Work with folders

1. Invoke `getStore()` on the default session.
   `Store store = Session.waitForDefaultSession().getStore();`
2. Complete any of the following tasks:

Task	Steps

Open a folder view.	1. Invoke `store.list()` to retrieve a list of folders.
	`Store store = null;`
	`store = Session.waitForDefaultSession().getStore();`
	`Folder[] folders = store.list();`
	2. Invoke `invokeApplication()` using the APP_TYPE_MESSAGES constant parameter and a new `MessageArguments` object that uses a folder from the list of folders as a parameter.
	`Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( folders[0]));`

List the folders in a mailbox store.	> Invoke `Store.list()`. `Folder[] folders = store.list();`

Retrieve an array of folders by type.	> Invoke `list(int)`. Provide the folder type as a parameter to this method.
	`Folder[] folders = store.lst(INBOX);`
	`Folder inbox = folders[0];`

Task	Steps
Retrieve an array of folders through a search.	> Invoke `findFolder(String)`. `Folder[] folders = store.findFolder("Inbox");`
Retrieve a folder by its name.	1. Invoke `getFolder(String)` and provide as a parameter the absolute path to the folder. `Folder folder = store.getFolder("Mailbox - Aisha Wahl/Inbox/Projects");` 2. Create code to manage a `FolderNotFoundException` exception if the folder does not exist.

Retrieve a folder by its ID.	1. Invoke `getID()` to retrieve the folder ID.
	`Folder[] folders = store.list();`
	`long id = folders[0].getId();`
	2. Invoke `getFolder()` with the ID as a parameter.
	`Folder f2 = store.getFolder(id);`

File a message.	> Invoke `appendMessage(Message)` on a `Folder` object.
	`Message msg = new Message();`
	`//...`
	`Folder folder = store.getFolder("Inbox");`
	`folder.appendMessage(msg);`

Working with attachments

To open incoming message attachments and create outgoing attachments on the BlackBerry® device, use the mail API. A separate `BodyPart` on a `Multipart` message represents a message attachment.

Create an attachment handler

The BlackBerry® Attachment Service receives all attachments first. Third-party attachment handlers cannot override the default BlackBerry device behavior. See the *BlackBerry Enterprise Server Maintenance and Troubleshooting Guide* for more information about the BlackBerry Attachment Service.

Task	Steps
Define a custom attachment handler.	> Implement the `AttachmentHandler` interface.

Register the accepted MIME types when the BlackBerry device receives an attachment.	> Implement `supports(String)`.
	`public boolean supports(String contentType) {`
	`return (contentType.toLowerCase().indexOf("contenttype") != -1 ? true : false);`
	`}`

Define the associated menu item string to display in the message list when the BlackBerry device user selects an attachment.	> Implement `menuString()`.
	`public String menuString() {`
	`return "Custom Attachment Viewer";`
	`}`

Define attachment processing.	> Implement `run()`. When a BlackBerry device user selects a menu item from the message list, this action invokes the `run()` method.
	`public void run(Message m, SupportedAttachmentPart p) {`
	`// Perform processing on data.`
	`Screen view = new Screen();`
	`view.setTitle(new LabelField("Attachment Viewer"));`
	`view.add(new RichTextField(new String((byte[])p.getContent())));`
	`}`

Register an attachment.	> Invoke `addAttachmentHandler()`.
	`AttachmentHandlerManager m = AttachmentHandlerManager.getInstance();`
	`CustomAttachmentHandler ah = new CustomAttachmentHandler();`
	`m.addAttachmentHandler(ah);`

Retrieve attachments

Task	Steps
Retrieve the contents of an attachment.	> Invoke `getContent()`. `String s = new String((byte[])p.getContent());`

Task	Steps

Retrieve information about the attachment.	> Invoke the methods of the `SupportedAttachmentPart` class. The `SupportedAttachmentPart` class represents an attachment with a corresponding viewer on the BlackBerry® device. An `UnsupportedAttachmentPart` represents an attachment that does not have a viewer on the BlackBerry device.
	`public void run(Message m, SupportedAttachmentPart p) {`
	`...`
	`String name = p.getName();`
	`int size = p.getSize();`
	`}`

Send a message with an attachment

Task	Steps

Create a multipart message.	> Create a new `Multipart` object.
	`byte[] buf = new byte[256]; // The attachment.`
	`MultiPart multipart = new MultiPart(); // Default type of multipart/mixed.`

Create each component of the attachment.	> Create a `SupportedAttachmentPart` object, designating the `Multipart` object as its parent. `SupportedAttachmentPart attach = new SupportedAttachmentPart( multipart, "application/x-example", "filename", data);`
Add each SupportedAttachmentPart object to the multipart object.	> Invoke `addBodyPart(SupportedAttachmentPart)` on that `object`. `multipart.addBodyPart(attach); // Add the attachment to the multipart.`
Set the content of the attachment.	> Invoke `setContent(Multipart)` on the `Message` object and pass in the `Multipart` object as its parameter. `msg.setContent(multipart);`
Send the message.	> Invoke `Transport.send()`. `Transport.send(msg);`

11

Using PIM applications

Using the calendar
Using the address book
Using tasks

Using the calendar

Start the calendar from your application

Task	Steps
Open the calendar.	> Invoke `Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments).`

View or change an event.	1. Retrieve an `Event` from the list of events.
	`Event e = null;`
	`EventList el = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE );`
	`Enumeration events = el.items();`
	`e = (Event)events.nextElement();`
	2. Invoke `Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)` using the `CalendarArguments` object created using the ARG_VIEW_DEFAULT property and the retrieved `Event`.
	`Invoke.invokeApplication( Invoke.APP_TYPE_CALENDAR, new CalendarArguments( CalendarArguments.ARG_VIEW_DEFAULT, e ) );`

Manage exceptions	> Check for a `ControlledAccessException` if your application invokes a BlackBerry® application that you do not have permission to use or access.

Task	Steps

Open a new populated event.	1. Create a new `Event` using an `EventList object`.
	`Event e = null;`
	`EventList el = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE );`
	`e = el.createEvent();`
	2. Add information to the `Event` object.
	`e.addString( Event.SUMMARY, 0, "Go For A Walk" );`
	`e.addString( Event.LOCATION, 0, "The Park" );`
	`long start = System.currentTimeMillis() + 8640000;`
	`e.addDate( Event.START, 0, start );`
	`e.addDate( Event.END, 0, start + 72000000 );`
	3. Invoke `Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)` using the `CalendarArguments` object created using the ARG_NEW property and the `Event`.
	`Invoke.invokeApplication( Invoke.APP_TYPE_CALENDAR, new CalendarArguments( CalendarArguments.ARG_NEW, e ) );`
	4. Use an instance of the `EventList` class to access the calendar.
	5. Create one or more `Event` object to store information for specific appointments. For each event, you can store data such as the summary, location, start and end times, and reminder notification.

Use the calendar

Task	Steps

Open an event list.	> Create an `EventList` object by invoking `openPIMList()`, providing as parameters the type of list to open (`PIM.EVENT_LIST`) and the mode in which to open the list:
	• READ_WRITE
	• READ_ONLY
	• WRITE_ONLY
	`EventList eventList = null;`
	`try {`
	`eventList = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE);`
	`} catch (PimException e) {`
	`// Handle exception.`
	`}`

Create an appointment.	> Invoke `createEvent()` on an event list. `Event event = eventList.createEvent():`

Task	Steps

Add appointment information.	> To verify that an item supports a field, invoke `isSupportedField(int)`.
	`if (event.isSupportedField(Event.SUMMARY)) {`
	`event.addString(Event.SUMMARY, Event.ATTR_NONE, "Meet with customer");`
	`}`
	`if (event.isSupportedField(Event.LOCATION)) {`
	`event.addString(Event.LOCATION, Event.ATTR_NONE, "Conference Center");`
	`}`
	`Date start = new Date(System.currentTimeMillis() + 8640000);`
	`if (event.isSupportedField(Event.START)) {`
	`event.addDate(Event.START, Event.ATTR_NONE, start);`
	`}`
	`if (event.isSupportedField(Event.END)) {`
	`event.addDate(Event.END, Event.ATTR_NONE, start + 72000000);`
	`}`
	`if (event.isSupportedField(Event.ALARM)) {`
	`if (event.countValues(Event.ALARM) > 0) {`
	`event.removeValue(Event.ALARM,0);`
	`event.setInt(Event.ALARM, 0, Event.ATTR_NONE, 396000);`
	`}`
	`}`

Create a recurring appointment.	1. Create a `RepeatRule` object. The `RepeatRule` class defines the fields for the properties and values that you can set, such as COUNT, FREQUENCY, and INTERVAL.
	2. To retrieve an array of supported fields, invoke `RepeatRule.getFields()`.
	3. To define a recurring pattern, invoke `setInt(int, int)` or `setDate(int, int, int, long)` on a new `RepeatRule` object.
	`RepeatRule recurring = new RepeatRule();`
	`recurring.setInt(RepeatRule.FREQUENCY, RepeatRule.MONTHLY);`
	`recurring.setInt(RepeatRule.DAY_IN_MONTH, 14);`
	4. To assign a recurrence pattern to an appointment, invoke `setRepeat(RepeatRule)` on an event.
	`EventList eventList = (EventList)PIM.getInstance().openPIMList(`
	`PIM.EVENT_LIST, PIM.READ_WRITE);`
	`Event event = eventList.createEvent();`
	`event.setRepeat(recurring);`

Change appointment information.	1. To replace an existing value with a new one, invoke the appropriate set method, such as `setString()`.
	2. To determine if a value is already set for the field, invoke `countValues()`.
	3. To change an existing value, use the corresponding set method, such as `setString()`.
	`if (event.countValues(Event.LOCATION) > 0) {`
	`event.setString(Event.LOCATION, 0, Event.ATTR_NONE, "Board Room");`
	`}`

Task	Steps

Save an appointment.	To save an appointment, use the `importEvent()` method; you do not have to invoke `commit()`.
	1. Before you save the appointment, to identify appointment fields that have changed since the appointment was last saved, invoke `isModified()`.
	2. Invoke `commit()`.
	`if(event.isModified()) {`
	`event.commit();`
	`}`

Retrieve appointment information.	1. To retrieve an enumeration of appointments, invoke `PIMList.items()`.
	`EventList eventList = (EventList)PIM.getInstance().openPIMList(`
	`PIM.EVENT_LIST, PIM.READ_ONLY);`
	`Enumeration e = eventList.items();`
	2. To retrieve an array of IDs of fields that have data for a particular task, invoke `PIMItem.getFields()`.
	3. To retrieve the field values, invoke `PIMItem.getString()`.
	`while (e.hasMoreElements()) {`
	`Event event = (Event)e.nextElement();`
	`int[] fieldIds = event.getFields();`
	`int id;`
	`for(int index = 0; index < fieldIds.length; ++index) {`
	`id = fieldIds[index];`
	`if(e.getPIMList().getFieldDataType(id) == STRING) {`
	`for(int j=0; j < event.countValues(id); ++j) {`
	`String value = event.getString(id, j);`
	`System.out.println(event.getFieldLable(id) + "=" + value);`
	`}`
	`}`
	`}`
	`}`

Export an appointment.	1. To import or export PIM data, use an output stream writer to export tasks from the BlackBerry device to a supported serial format, such as iCal®.
	2. To retrieve a string array of supported serial formats, invoke `PIM.supportedSerialFormats()`, and then specify the list type (`PIM.EVENT_List`).
	3. To write an item in serial format, invoke `toSerialFormat()`. The **enc** parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null.
	`EventList eventList = (EventList)PIM.getInstance().openPIMList(`
	`PIM.EVENT_LIST, PIM.READ_ONLY );`
	`ByteArrayOutputStream bytestream = new ByteArrayOutputStream();`
	`Enumeration e = eventList.items();`
	`while (e.hasMoreElements()) {`
	`Event event = (Event)e.nextElement();`
	`PIM.getInstance().toSerialFormat(event, byteStream, "UTF8",`
	`dataFormats[0]);`
	`}`

Task	Steps

Import an appointment.	1. `Write appointment to iCal.`
	`String[] dataFormats = PIM.eventSerialFormats();`
	`ByteArrayOutputStream os = new ByteArrayOutputStream();`
	`PIM.getInstance().toSerialFormat(event, os, "UTF8", dataFormats[0]);`
	2. `Import appointment from iCal.`
	`ByteArrayInputStream is = new ByteArrayInputStream(outputStream.toByteArray());`
	3. To return an array of `PIMItem` objects, invoke `fromSerialFormat(java.io.InputStream is, java.lang.String enc)`
	`PIMItem[] pi = PIM.getInstance().fromSerialFormat(is, "UTF8");`
	4. To add a new appointment, invoke `EventList.importEvent()`.
	`EventList eventList = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE);`
	`Event event2 = eventList.importEvent((Event)pi[0]);`

Close an event list.	1. Invoke `close()`.
	2. Create a try-catch block to manage a `PimException`.
	`try {`
	`eventList.close();`
	`} catch (PimException e) {`
	`// Handle exception.`
	`}`

Code sample: Creating new recurring appointments

To let the BlackBerry® device user invite attendees to the meeting, combine this code sample with ContactsDemo.java.

See "Code sample: Displaying a screen that lets BlackBerry device users add new contacts" on page 175 for more information.

Example: EventDemo.java

```
/**
 * EventDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.eventdemo;
import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public final class EventDemo extends UiApplication
{
```

```
    private EventScreen _eventScreen;

    public static void main(String[] args)  {
        new EventDemo().enterEventDispatcher();
    }

    private EventDemo()     {
        _eventScreen = new EventScreen();
        pushScreen(_eventScreen);
    }

    public final static class EventScreen extends MainScreen
    {
        private EditField _subject, _location;
        private SaveMenuItem _saveMenuItem;
        private DateField _startTime, _endTime;
        private ObjectChoiceField _repeat;
        private Event event;

        private class SaveMenuItem extends MenuItem {
            public SaveMenuItem() {
                super(null, 0, 100000, 5);
            }

            public String toString() {
                return “Save”;
            }

            public void run() {
                onSave();
            }
        }

        public EventScreen() {
            _saveMenuItem = new SaveMenuItem();
            setTitle(new LabelField(“Event Demo”, LabelField.ELLIPSIS |
                LabelField.USE_ALL_WIDTH) );
            _subject = new EditField(“Subject: “, ““);
            add(_subject);
            _location = new EditField(“Location: “, ““);
            add(_location);
            _startTime = new DateField(“Start: “, System.currentTimeMillis() +
                3600000, DateField.DATE_TIME);
            _endTime = new DateField(“End: “, System.currentTimeMillis() +
                7200000, DateField.DATE_TIME);
            add(new SeparatorField());
            add(_startTime);
            add(_endTime);
            add(new SeparatorField());
            String[] choices = {“None”, “Daily”, “Weekly”, “Monthly”, “Yearly”};
            _repeat = new ObjectChoiceField(“Recurrence: “, choices, 0);
            add(_repeat);
        }

        protected boolean onSave() {
            try {
                EventList eventList = (EventList)PIM.getInstance().
                    openPIMList(PIM.EVENT_LIST, PIM.WRITE_ONLY);
```

```
                event = eventList.createEvent();
                event.addString(Event.SUMMARY, PIMItem.ATTR_NONE,
                    _subject.getText());
                event.addString(Event.LOCATION, PIMItem.ATTR_NONE,
                    _location.getText());
                event.addDate(Event.END, PIMItem.ATTR_NONE, _endTime.getDate());
                event.addDate(Event.START, PIMItem.ATTR_NONE,
                    _startTime.getDate());
                if(_repeat.getSelectedIndex() != 0) {
                    event.setRepeat(setRule());
                }
                // Save the appointment to the Calendar.
                event.commit();
                //reset fields on screen
                _subject.setText("");
                _location.setText("");
                _endTime.setDate(null);
                _startTime.setDate(null);
                _repeat.setSelectedIndex(0);
                return true;
            } catch (PIMException e) {
                System.err.println(e);
            }
            return false;
        }

        private RepeatRule setRule() {
            RepeatRule rule = new RepeatRule();
            int index = _repeat.getSelectedIndex();
                if (index == 0) {
                    rule.setInt(RepeatRule.FREQUENCY, RepeatRule.DAILY);
                }
                if (index == 1) {
                    rule.setInt(RepeatRule.FREQUENCY, RepeatRule.WEEKLY);
                }
                if (index == 2) {
                    rule.setInt(RepeatRule.FREQUENCY, RepeatRule.MONTHLY);
                }
                if (index == 3) {
                    rule.setInt(RepeatRule.FREQUENCY, RepeatRule.YEARLY);
                }
                return rule;
        }

        protected void makeMenu(Menu menu, int instance)               {
            menu.add(_saveMenuItem);
            menu.addSeparator();
            super.makeMenu(menu, instance);
        }
    }
}
```

Using the address book

Open the address book from your application

Task	Steps
Open the address book.	> From an application, invoke `Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK,AddressBookArguments).`

Open a contact using PIM data.	1. Create an instance of an `AddressBookArguments` object, specifying as a parameter a `Contact` object.
	`AddressBookArguments abArg = AddressBookArguments(String arg, Contact contact);`
	2. Invoke `Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, AddressBookArguments)` using the `AddressBookArguments` object for the contact.
	`Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, abArg);`

Manage exceptions.	> Check for a `ControlledAccessException` if your application invokes a BlackBerry® application that you do not have permission to use or access.

Use contacts

Task	Steps
Provide access to the PIN BlackBerry® device contacts field.	> Use the `BlackBerryContact.PIN` constant.
Provide access to the USER1 through USER4 BlackBerry device contacts fields.	> Use the following constants: • `BlackBerryContact.USER1` • `BlackBerryContact.USER2` • `BlackBerryContact.USER3` • `BlackBerryContact.USER4`
Define labels for the USER1 through USER4 BlackBerry device contacts fields.	Changing a label affects all contacts on the BlackBerry device. > Invoke `BlackBerryPIMList.setFieldLabel().`

Open a contacts list.	1. Create a contacts list.
	`ContactList contactList = null;`
	2. Invoke `PIM.openPIMList()` and provide as parameters the type of list to open `(PIM.CONTACT_LIST)` and the access mode with which to open the list (READ_WRITE, READ_ONLY, or WRITE_ONLY).
	`try {`
	`contactList = (ContactList)PIM.getInstance().openPIMList(`
	`PIM.CONTACT_LIST, PIM.READ_WRITE);`
	`} catch (PimException e) {`
	`return;`
	`}`

Task	Steps
Create a contact.	To add a contact to the database, you must commit it. See "Save a contact" on page 173 for more information about committing contact data. > Invoke `createContact()` on a contacts list. `Contact contact = contactList.createContact();`

Task	Steps

Add contact information.	1. Invoke one of the following methods:
	• `addString()`
	• `addStringArray()`
	• `addDate()`
	• `addInt()`
	• `addBoolean()`
	• `addBinary()`
	2. Before you set or retrieve a field, to verify that the item supports the field, invoke `ContactList.isSupportedField(int)`.
	3. To let fields store multiple values, use field attributes. For example, the TEL field supports the ATTR_HOME, ATTR_WORK, ATTR_MOBILE, and ATTR_FAX attributes to store numbers for work, home, mobile, and fax numbers.
	4. To determine how many values a field supports, invoke `PIMList.maxValues(int field)`.
	5. To verify that a field supports a particular attribute, invoke `isSupportedAttribute(int, int)`.
	`// Create string array for name.`
	`try {ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.WRITE_ONLY);}`
	`catch (PIMException e) {}`
	`Contact contact = contactList.createContact();String[] name = new String[5];`
	`// 5 name elements`
	`try {name[Contact.NAME_PREFIX] = "Mr.";name[Contact.NAME_FAMILY] = "McPherson";name[Contact.NAME_GIVEN] = "Scott";}`
	`catch (IllegalArgumentException iae)`
	`{// handle exception}`
	`// Add name.`
	`if(contactList.isSupportedField(Contact.NAME)) {contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name);`
	`}`
	`// Create string array for address.`
	`String[] address = new String[7];`
	`// 7 address elements`
	`try {address[Contact.ADDR_COUNTRY] = "United States";address[Contact.ADDR_LOCALITY] = "Los Angeles";address[Contact.ADDR_POSTALCODE] = "632300";address[Contact.ADDR_REGION] = "California";address[Contact.ADDR_STREET] = "323 Main Street";}`
	`catch (IllegalArgumentException iae) {// Handle exception.}`
	`// Add address.contact.addStringArray(Contact.ADDR, Contact.ATTR_NONE, address);`
	`// Add home telephone number.`
	`if (contactList.isSupportedField(Contact.TEL) &&contactList.isSupportedAttribute(Contact.TEL, Contact.ATTR_HOME)) {contact.addString(Contact.TEL, Contact.ATTR_HOME, "555-1234");}`
	`// Add work telephone number.if (contactList.isSupportedField(Contact.TEL)) {contact.addString(Contact.TEL, Contact.ATTR_HOME, "555-5555");}`
	`// Add work internet messaging address.`
	`if (contactList.isSupportedField(Contact.EMAIL)) {contact.addString(Contact.EMAIL, Contact.ATTR_WORK, "aisha.wahl@blackberry.com");}`

Task	Steps

Change contact information.	1. To change the name and address fields, invoke the appropriate set method to replace an existing value with a new value.
	2. Perform one of the following actions:
	• To change the fields that support a single value, retrieve the array and then change one or more indexes in the array before adding the array back to the `Contact` object.
	`if (contact.countValues(Contact.NAME) > 0) {`
	`String[] newname = contact.getStringArray(Contact.NAME, 0);`
	`}`
	`// Change the prefix to Dr. and add the suffix, Jr.`
	`newname[Contact.NAME_PREFIX] = "Dr.";`
	`newname[Contact.NAME_SUFFIX] = "Jr.";`
	`contact.setStringArray(Contact.NAME, 0, Contact.ATTR_NONE, newname);`
	• To change the contacts fields that support multiple values, before adding another value, verify that the number of values does not exceed the maximum number of values. For example:
	`if (contact.countValues(Contact.EMAIL) < contactList.maxValues(Contact.EMAIL)) {`
	`contact.addString(Contact.EMAIL, Contact.ATTR_NONE, "aisha.wahl@blackberry.com");}`
	3. Create code to manage a `FieldFullException`, which occurs if you invoke an `add` method, such as `addString()`, for a field that already has a value.

Save a contact.	1. To determine if any contact fields have changed since the contact was last saved, invoke `isModified()`.
	2. Invoke `commit()`.
	`if(contact.isModified()) {`
	`contact.commit();`
	`}`

Task	Steps
Retrieve contact information.	1. Invoke `PIMList.items()`. 2. Perform one of the following actions: • To retrieve an array of IDs for fields that have data for a particular contact, invoke `PIMItem.getFields()` . • To retrieve the field values, invoke `PIMItem.getString()`. 3. When you invoke `PIMList.items()` to retrieve an enumeration of items in a contacts list, your application must sort items as necessary. `ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE); Enumeration enum = contactList.items(); while (enum.hasMoreElements()) { Contact c = (Contact)enum.nextElement(); int[] fieldIds = c.getFields(); int id; for(int index = 0; index < fieldIds.length; ++index) { id = fieldIds[index]; if(c.getPIMList().getFieldDataType(id) == Contact.STRING) { for(int j=0; j < c.countValues(id); ++j) { String value = c.getString(id, j); System.out.println(c.getPIMList().getFieldLabel(id) + "=" + value); } } } }`
Export a contact.	1. To import or export PIM data, use an output stream writer to export tasks from the BlackBerry device to a supported serial format, such as vCard®. 2. To retrieve a string array of supported formats, invoke `PIM.supportedSerialFormats()` and specify the list type `(PIM.Contact_LIST)`. 3. To write an item to a supported serial format, invoke `toSerialFormat()`. The **enc** parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null. `ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_ONLY); String[] dataFormats = PIM.getInstance().supportedSerialFormats(PIM.CONTACT_LIST); ByteArrayOutputStream byteStream = new ByteArrayOutputStream(); Enumeration e = contactList.items(); while (e.hasMoreElements()) { Contact c = (Contact)e.nextElement(); PIM.getInstance().toSerialFormat(c, byteStream, "UTF8", dataFormats[0]); }`

Task	Steps

Import a contact.	1. To return an array of PIM items, invoke `fromSerialFormat()`.
	`ByteArrayInputStream is = new ByteArrayInputStream(outputStream.toByteArray());`
	`PIMItem[] pi = PIM.getInstance().fromSerialFormat(istream, "UTF8");`
	2. To create a new contact using the PIM item, invoke `ContactList.importContact();`
	`ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE);`
	`Contact contact2 = contactList.importContact((Contact)pi[0]);`
	`contact2.commit();`
	3. To specify the character encoding to use when writing to the output stream, use the enc parameter.

Delete a contact.	> Invoke `removeContact()` on a contacts list. `contactList.removeContact(contact);`

Close a contacts list.	> Invoke `close()`.
	`try {`
	`contactList.close();`
	`} catch(PIMException e) {`
	`Dialog.alert(e.toString());`
	`}`

Code sample: Displaying a screen that lets BlackBerry device users add new contacts

The following code sample demonstrates how to display a screen that lets BlackBerry® device users add new contacts to their address books.

Example: ContactsDemo.java

```
/**
 * ContactsDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.contactsdemo;

import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import net.rim.blackberry.api.pdap.*;

public final class ContactsDemo extends UiApplication
```

```
{
    private ContactScreen _contactScreen;

    public static void main(String[] args) {
        new ContactsDemo().enterEventDispatcher();
    }

    public ContactsDemo() {
        _contactScreen = new ContactScreen();
        pushScreen(_contactScreen);
    }

    // Inner class. Creates a Screen to add a contact.
    public static final class ContactScreen extends MainScreen
    {
        private EditField _first, _last, _email, _phone, _pin;
        private SaveMenuItem _saveMenuItem;
        private class SaveMenuItem extends MenuItem {
            private SaveMenuItem() {
                super(null, 0, 100000, 5);
            }
            public String toString() {
                return “Save”;
            }
            public void run() {
                onSave();
            }
        }

        public ContactScreen() {
            _saveMenuItem = new SaveMenuItem();
            setTitle(new LabelField(“Contacts Demo”, LabelField.ELLIPSIS |
LabelField.USE_ALL_WIDTH));
            _first = new EditField(“First Name: “, ““);
            add(_first);
            _last = new EditField(“Last Name: “, ““);
            add(_last);
            _email = new EditField(“Email Address: “, ““,
            BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_EMAIL);
            add(_email);
            _phone = new EditField(“Work Phone: “, ““,
            BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_PHONE);
            add(_phone);
            _pin = new EditField(“PIN:”, ““, 8, BasicEditField.FILTER_HEXADECIMAL);
            add(_pin);
        }

        protected boolean onSave() {
            String firstName = _first.getText();
            String lastName = _last.getText();
            String email = _email.getText();
            String phone = _phone.getText();
            String pin = _pin.getText();
            // Verify that a first or last name and email has been entered.
            if ((firstName.equals(““) && lastName.equals(““)) || email.equals(““)) {
                Dialog.inform(“You must enter a name and an email address!”);
                return false;
            } else {
```

```
                try {
                    ContactList contactList =
(ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.WRITE_ONLY);
                    Contact contact = contactList.createContact();
                    String[] name = new String[contactList.stringArraySize(Contact.NAME)];
                    // Add values to PIM item.
                    if (!firstName.equals("")) {
                        name[Contact.NAME_GIVEN] = firstName;
                    }
                    if (!lastName.equals("")) {
                        name[Contact.NAME_FAMILY] = lastName;
                    }
                    contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name);
                    contact.addString(Contact.EMAIL, Contact.ATTR_HOME, email);
                    contact.addString(Contact.TEL, Contact.ATTR_WORK, phone);
                    if (contactList.isSupportedField(BlackBerryContact.PIN)) {
                        contact.addString(BlackBerryContact.PIN, Contact.ATTR_NONE, pin);
                    }
                    // Save data to address book.
                    contact.commit();
                    // Reset UI fields.
                    _first.setText("");
                    _last.setText("");
                    _email.setText("");
                    _phone.setText("");
                    _pin.setText("");
                    return true;
                } catch (PIMException e) {
                    return false;
                }
            }
        }

        protected void makeMenu(Menu menu, int instance) {
            menu.add(_saveMenuItem);
            super.makeMenu(menu, instance);
        }
    }
}
```

Using tasks

Start the task application from your application

Check for a `ControlledAccessException` if your application invokes a BlackBerry® application that you do not have permission to use or access.

Task	Steps
Open the task application.	The `TaskArguments` (net.rim.blackberry.api.invoke.TaskArguments) cannot be updated without changes to the task application. > Invoke `Invoke.invokeApplication(APP_TYPE_TASKS, TaskArguments).`

View or change a task.	1. Create an instance of a `ToDoList` and store it in an `Enumeration`.
	`ToDoList tdl = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE);`
	`Enumeration todos = tdl.items();`
	2. Create a `ToDo` object using an element from the `Enumeration`:
	`ToDo todo = (ToDo)todos.nextElement();`
	3. Invoke `invokeApplication()` using the APP_TYPE_TASKS constant parameter, and a new `TaskArguments` object created using the ARG_VIEW parameter and the `ToDo` object.
	`Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_VIEW, todo));`

Create a new blank task.	> Invoke `invokeApplication()` using the APP_TYPE_TASKS constant parameter, and a new `TaskArguments` object created using the ARG_NEW parameter. `Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments( TaskArguments.ARG_NEW) );`

Create a new populated task.	1. Create an instance of a `ToDoList`.
	`ToDoList tdl = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE);`
	2. Invoke `createToDo()` to create a new `ToDo` object and add information to the new `ToDo` object.
	`ToDo todo = tdl.createToDo();`
	`todo.addString(ToDo.SUMMARY, 0, "Walk the Dog");`
	3. Invoke `invokeApplication()` using the APP_TYPE_TASKS constant parameter, and a new `TaskArguments` object created using the ARG_NEW parameter and the new `ToDo` object.
	`Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments( TaskArguments.ARG_NEW, todo));`

Use tasks

Task	Steps

Open a task list.	> Invoke `PIM.openPIMList()` and provide as parameters the type of list to open (`PIM.TODO_LIST`) and the access mode with which to open the list (READ_WRITE, READ_ONLY, or WRITE_ONLY).
	`ToDoList todoList = null;`
	`try {`
	`todoList = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE);`
	`} catch (PimException e) {`
	`//an error occurred`
	`return;`
	`}`

Create a task.	> Invoke `createToDo()` on a task list. `ToDo task = todoList.createToDo();`

Add task information.	1. Before you set or retrieve a field, verify that the item supports the field by invoking `isSupportedField(int)`.
	2. To retrieve the field data type, invoke `PIMList.getFieldDataType(int)`.
	3. To set the field data, invoke one of the following methods:
	• `addString()`
	• `addDate()`
	• `addInt()`
	• `addBoolean()`
	• `addBinary()`
	`if (todoList.isSupportedField(ToDo.SUMMARY)) {`
	`task.addString(ToDo.SUMMARY, ToDo.ATTR_NONE, "Create project plan");`
	`}`
	`if (todoList.isSupportedField(ToDo.DUE)) {`
	`Date date = new Date();`
	`task.addDate(ToDo.DUE, ToDo.ATTR_NONE, (date + 17280000));`
	`}`
	`if (todoList.isSupportedField(ToDo.NOTE)) {`
	`task.addString(ToDo.NOTE, ToDo.ATTR_NONE, "Required for meeting");`
	`}`
	`if (todoList.isSupportedField(ToDo.PRIORITY)) {`
	`task.addInt(Todo.PRIORITY, ToDo.ATTR_NONE, 2);`
	`}`

Set the status of a task.	> `Use the PIM extended field ToDo.EXTENDED_FIELD_MIN_VALUE + 9:` • `STATUS_NOT_STARTED: 1` • `STATUS_IN_PROGRESS: 2` • `STATUS_COMPLETED: 3` • `STATUS_WAITING: 4` `task.addInt(ToDo.EXTENDED_FIELD_MIN_VALUE + 9, ToDo.ATTR_NONE, 2);`

Task	Steps

Change task information.	1. To replace an existing value with a new value, invoke the appropriate set method, such as `setString()`.
	2. To determine if a value is already set for the field, invoke `countValues()`.
	3. To change an existing value, use the corresponding `set()` method.
	4. Create code to manage a `FieldFullException`, which a method such as `addString()` throws when a value already exists.
	`if (task.countValues(ToDo.SUMMARY) > 0) {`
	`task.setString(ToDo.SUMMARY, 0, ToDo.ATTR_NONE, "Review notes");`
	`}`

Save a task.	1. Before you commit the changes, to determine whether any task fields have changed since the task was last saved, invoke `isModified()`.
	2. Invoke `commit()`.
	`if(task.isModified()) {`
	`task.commit();`
	`}`

Retrieve task information.	1. To retrieve an enumeration, invoke `PIMList.items()` on the task list.
	`ToDoList todoList = (ToDoList)PIM.getInstance().openToDoList(`
	`PIM.TODO_LIST, PIM.READ_ONLY);`
	`Enumeration enum = todoList.items();`
	2. To retrieve an array of IDs for fields that have data for a particular `ToDo` item, invoke `PIMItem.getFields()`.
	3. To retrieve the field values, invoke `PIMItem.getString()`.
	`while (enum.hasMoreElements()) {`
	`ToDo task = (ToDo)enum.nextElement();`
	`int[] fieldIds = task.getFields();`
	`int id;`
	`for(int index = 0; index < fieldIds.length; ++index) {`
	`id = fieldIds[index];`
	`if(task.getPIMList().getFieldDataType(id) == STRING) {`
	`for(int j=0; j < task.countValues(id); ++j) {`
	`String value = task.getString(id, j);`
	`System.out.println(task.getFieldLable(id) + "=" + value);`
	`}`
	`}`
	`}`
	`}`

Task	Steps

Export a task.	1. To import or export PIM data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format.
	`ToDoList todoList = (ToDoList)PIM.getInstance().openPIMList( PIM.TODO_LIST, PIM.READ_ONLY);`
	`ByteArrayOutputStream byteStream = new ByteArrayOutputStream();`
	2. To retrieve a string array of supported serial formats, invoke `PIM.supportedSerialFormats()`, and then specify the list type (`PIM.TODO_List`).
	`String[] dataFormats = PIM.getInstance().supportedSerialFormats(PIM.TODO_LIST);`
	3. To write an item to a serial format, invoke `toSerialFormat()`. The **enc** parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null.
	`Enumeration e = todoList.items();`
	`while (e.hasMoreElements()) {`
	`ToDo task = (ToDo)e.nextElement();`
	`PIM.getInstance().toSerialFormat(task, byteStream, "UTF8", dataFormats[0]);`
	`}`

Import a task.	1. To return an array of `PIMItem` objects, invoke `fromSerialFormat()`. The **enc** parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null.
	`String[] dataFormats = PIM.toDoSerialFormats();`
	`// Write task to serial format.`
	`ByteArrayOutputStream os = new ByteArrayOutputStream();`
	`PIM.getInstance().toSerialFormat(task, os, "UTF8", dataFormats[0]);`
	`// Import task from serial format.`
	`ByteArrayInputStream is = new ByteArrayInputStream(outputStream.toByteArray());`
	`PIMItem[] pi = PIM.getInstance().fromSerialFormat(is, "UTF8");`
	2. To create a new task using the PIM data, invoke `ToDoList.importToDo()`. The `importToDo()` method saves the task; you do not have to invoke `commit()`.
	`ToDoList todoList = (ToDoList)PIM.getInstance().openPIMList( PIM.TODO_LIST, PIM.READ_WRITE);`
	`ToDo task2 = todoList.importToDo((ToDo)pi[0]);`

Delete a task.	> Invoke `removeToDo()` on a task list. `todoList.removeToDo(task);`

Close a task list.	1. Invoke `todoList.close()`.
	2. Create code that manages exceptions.
	`try {`
	`todoList.close();`
	`} catch (PimException e) {`
	`// Handle exception.`
	`}`

Code sample:

Example: TaskDemo.java

```
/**
 * TaskDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.taskdemo;
import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public final class TaskDemo extends UiApplication
{
    private TaskScreen _taskScreen;

    public static void main(String[] args) {
        new TaskDemo().enterEventDispatcher();
    }

    private TaskDemo() {
        _taskScreen = new TaskScreen();
        pushScreen(_taskScreen);
    }

    public final static class TaskScreen extends MainScreen
    {
        // Members.
        private EditField _summary, _note;
        private DateField _due;
        private ObjectChoiceField _priority, _status;
        private SaveMenuItem _saveMenuItem;

        private class SaveMenuItem extends MenuItem
        {
            private SaveMenuItem() {
                super(null, 0, 100000, 5);
            }

            public String toString() {
                return “Save”;
            }

            public void run() {
                onSave();
            }
        }

        public TaskScreen() {
```

```
            _saveMenuItem = new SaveMenuItem();
            setTitle(new LabelField(“Tasks Demo”,
            LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH));
            _summary = new EditField(“Task Summary: “, ““);
            add(_summary);
            // In TODO.Priority, 0 to 9 is highest to lowest priority.
            String[] choices = {“High”, “Normal”, “Low”};
            _priority = new ObjectChoiceField(“Priority: “, choices, 1);
            add(_priority);
            String[] status = { “Not Started”, “In Progress”, “Completed”,
            “Waiting on someone else”, “Deferred” };
            _status = new ObjectChoiceField(“Status: “, status, 0);
            add(_status);
            _due = new DateField(“Due: “, System.currentTimeMillis() + 3600000,
            DateField.DATE_TIME);
            add(_due);
            _note = new EditField(“Extra Notes: “, ““);
            add(_note);
        }

        protected boolean onSave() {
            try {
                ToDoList todoList = (ToDoList)PIM.getInstance().
                openPIMList(PIM.TODO_LIST, PIM.WRITE_ONLY);
                ToDo task = todoList.createToDo();
                task.addDate(ToDo.DUE, ToDo.ATTR_NONE, _due.getDate());
                task.addString(ToDo.SUMMARY, ToDo.ATTR_NONE, _summary.getText());
                task.addString(ToDo.NOTE, ToDo.ATTR_NONE, _note.getText());
                task.addInt(ToDo.PRIORITY, ToDo.ATTR_NONE,
                _priority.getSelectedIndex());
                // ToDo.EXTENDED_FIELD_MIN_VALUE + 9 represents status.
                // Add 1 to selected index so that values are correct.
                // See the RIM Implementation Notes in the API docmentation for ToDo.
                task.addInt(ToDo.EXTENDED_FIELD_MIN_VALUE + 9, ToDo.ATTR_NONE,
                _status.getSelectedIndex() + 1);
                // Save task to handheld tasks.
                task.commit();
                _summary.setText(““);
                _note.setText(““);
                _due.setDate(null);
                _priority.setSelectedIndex(1); // Reset to “Normal” priority.
                _status.setSelectedIndex(0); // Reset to “Not Started” status.
                return true;
            } catch (PIMException e) {
                return false;
            }
        }

        protected void makeMenu(Menu menu, int instance) {
            menu.add(_saveMenuItem);
            super.makeMenu(menu, instance);
        }
    }
}
```

12

Using the phone application

Start the phone application from your application
Use phone call functionality
Listen for phone events
Access and use call logs

Start the phone application from your application

To open the phone application from your application, invoke `Invoke.invokeApplication(APP_TYPE_PHONE,PhoneArguments).`

The following excerpt from the Restaurants.java code sample on page 69 creates a menu item that invokes the phone application to call a restaurant.

```
private MenuItem phoneItem = new MenuItem(_resources.getString(MENUITEM_PHONE), 110, 12) {
public void run() {
synchronized(store) {
String phoneNumber = phonefield.getText();
if ( phoneNumber.length == 0 ) {
Dialog.alert(_resources.getString(ALERT_NO_PHONENUMBER));
} else {
PhoneArguments call = new PhoneArguments(PhoneArguments.ARG_CALL, phoneNumber);
Invoke.invokeApplication(Invoke.APP_TYPE_PHONE, call);
}
}
}
};
```

Use phone call functionality

Task	Steps
Retrieve a phone call.	> Invoke `Phone.getActiveCall()`. `PhoneCall call = Phone.getActiveCall();`
Retrieve a phone call by call ID.	> Invoke `Phone.getCall(int)`.

Task	Steps

Retrieve phone call information.	> Use the methods of the `PhoneCall` class.
	`int threshold = 120; // Alert user if outgoing calls last longer than threshold.`
	`int elapsedTime = call.getElapsedTime();`
	`// Use getStatusString() to retrieve status as an string.`
	`int status = call.getStatus();`
	`if ((status == PhoneCall.STATUS_CONNECTED || status == PhoneCall.STATUS_CONNECTING) && call.isOutGoing() && elapsedTime > threshold) {`
	`// Use getCallId() to retrieve the caller ID as as an integer.`
	`String phoneNumber = call.getDisplayPhoneNumber();`
	`Status.show("Your call to " + phoneNumber + " has lasted more than " + (String)threshold + ".");`
	`}`

Add DTMF tones to the send queue

Task	Steps
Add a single DTMF tone to the send queue.	> Invoke `sendDTMFTone()`.
Add multiple DTMF tones to the send queue.	> Invoke `sendDTMFTones()`.
Retrieve the send queue for the current call.	> Invoke `getDTMFTones()`.

BlackBerry DTMF tones

BlackBerry® devices play DTMF tones as soon as no other tones are pending.

DTMF tones consist of a low and a high frequency, which are played at the same time.

Key	Low Tone (Hz)	High Tone (Hz)
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
0	941	1209
*	941	1336
#	941	1477

Listen for phone events

Task	Steps
Listen for phone events.	> Implement the `PhoneListener` interface.
Register the phone listener.	> Invoke `Phone.addPhoneListener()`.
Remove a phone listener.	> Invoke `removePhoneListener()`.

To act on a particular event, implement one of the following methods.:

Event	Method
A call is added to a conference call.	`callAdded(int)`
A BlackBerry® device user answers a call (user driven).	`callAnswered(int)`
A conference call is established.	`callConferenceCallEstablished(int)`
The network indicates a connected event (network driven).	`callConnected(int)`
A direct-connect call is connected.	`callDirectConnectConnected(int)`
A direct-connect call is disconnected.	`callDirectConnectDisconnected(int)`
A call is disconnected.	`callDisconnected(int)`
A BlackBerry device user ends a call.	`callEndedByUser(int)`
A call fails.	`callFailed(int, int)`
A call goes on hold.	`callHeld(int)`
A new call arrives.	`callIncoming(int)`
The BlackBerry device initiates an outgoing call.	`callInitiated(int)`
A call is removed from a conference call.	`callRemoved(int)`
A held call resumes.	`callResumed(int)`
A call is waiting.	`callWaiting(int)`
A conference call ends (all members are disconnected).	`conferenceCallDisconnected(int)`

Access and use call logs

Task	Steps
Retrieve a phone log.	The `PhoneLogs` class represents the phone call history and provides methods for opening, adding, deleting, or swapping call logs. > Invoke `PhoneLogs.getInstance()`. `PhoneLogs _logs = PhoneLogs.getInstance();`
Retrieve the number of normal or missed calls.	Two phone log folders exist: FOLDER_NORMAL_CALLS and FOLDER_MISSED_CALLS. > Invoke `numberOfCalls(int)`. `int numberOfCalls = _logs.numberOfCalls(FOLDER_NORMAL_CALLS);`
Retrieve a call log.	You can instantiate two types of call logs: `PhoneCallLog` objects, which have only one participant, and `ConferencePhoneCallLog` objects, which have two or more participants. > Invoke `PhoneLogs.callAt(int index, long folderID)`. `PhoneCallLog phoneLog = (PhoneCallLog)_logs.callAt(0);`

Task	Steps

Retrieve a call participant by phone number.	The `PhoneCallLogID` class identifies participants in a phone call log by phone number.
	> Invoke `PhoneCallLog.getParticipant(int)` or `ConferencePhoneCallLog.getParticipantAt()`.
	`PhoneCallLogID participant = phoneCallLog.getParticipant();`
	`PhoneCallLogID participant = ConferencePhoneCallLog.getParticipant();`

Retrieve the phone number type.	The `PhoneCallLogID` class identifies the type of phone call for a log. For example, home, mobile, work, or fax, as recorded in the address book. > Invoke `PhoneCallLogID.getType()`. `String phoneType = PhoneCallLogID.getType();`

Create a call log or conference call log.	The `PhoneCallLogID` constructor removes dashes and other non-numeric characters from phone numbers.
	1. Create an instance of a `PhoneCallLog` or `ConferencePhoneCallLog` object, and provide the date, duration, participants, and notes for the call as parameters to the constructor.
	`Date date = new Date("1000"); // date of call`
	`int duration = 60; // duration of call`
	`PhoneCallLogID caller1 = new PhoneCallLogID("555-1234"); // first participant`
	`PhoneCallLogID caller2 = new PhoneCallLogID("555-1235"); // second participant`
	`String notes = "New call."; // notes`
	`ConferencePhoneCallLog conferenceCall = new ConferencePhoneCallLog(date, duration, PhoneLogs.FOLDER_NORMAL_CALLS, caller1, caller2, notes);`
	2. Update the call log:
	• To update the call log, invoke `PhoneLogs.addCall(CallLog call)`.
	`_logs.addCall(conferenceCall);`
	• To replace the call log with a new call log, invoke `PhoneLogs.swapCall(CallLog call,int index,long folderID)`.
	`_logs.swapCall(conferenceCall, 0, FOLDER_NORMAL_CALLS);`

Delete a call log.	> Invoke `PhoneLogs.deleteCall()`. `_logs.deleteCall(0);`

Code sample: Calculating the time that a participant spends on the phone

Example: PhoneLogsDemo.java

```
/**
 * PhoneLogsDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.phonelogs;

import net.rim.blackberry.api.phone.phonelogs.*;
import java.lang.*;
import net.rim.device.api.system.Application;
```

```
public class PhoneLogsDemo extends Application
{
    private PhoneLogs _logs;
    private int _timeSpokenTo;

    static public void main(String[] args) {
        PhoneLogsDemo app = new PhoneLogsDemo();
        app.enterEventDispatcher();
    }

    private PhoneLogsDemo() {
        _logs = PhoneLogs.getInstance();
        PhoneCallLogID participant = new PhoneCallLogID(“5551234”);
        _timeSpokenTo = findTimeSpokenTo(participant,
                            PhoneLogs.FOLDER_NORMAL_CALLS);
    }

    // Returns the number of seconds spent on the phone with a participant.
    public int findTimeSpokenTo(PhoneCallLogID participant,
        long folder) {
        int numberOfCalls = this._logs.numberOfCalls(folder);
        int timeSpokenTo = 0;
        PhoneCallLog phoneCallLog;
        ConferencePhoneCallLog conferencePhoneCallLog;
        for (int i = 0; i < numberOfCalls; i++) {
            Object o = _logs.callAt(i, folder);
            if (o instanceof PhoneCallLog) {
                phoneCallLog = (PhoneCallLog) o;
                if ( phoneCallLog.getParticipant() == participant)
                    timeSpokenTo += phoneCallLog.getDuration();
            } else {
                conferencePhoneCallLog = (ConferencePhoneCallLog) o;
                int participants = conferencePhoneCallLog.numberOfParticipants();
                for (int j = 0; j < participants; j++)
                    if (conferencePhoneCallLog.getParticipantAt(j) == participant) {
                        timeSpokenTo += conferencePhoneCallLog.getDuration();
                        j = participants;
                    }
            }
        }
        return timeSpokenTo;
    }
}
```

13

Using the BlackBerry Browser

Display content in the BlackBerry Browser
Display content in a BlackBerry Browser field

Display content in the BlackBerry Browser

To display web content in the BlackBerry® Browser, use the `net.rim.blackberry.api.browser` package.

Task	Steps
Retrieve a BlackBerry Browser session.	Retrieving the default session overrides any open sessions on the BlackBerry device. > Retrieve the default `BrowserSession` object by invoking the static method `Browser.getDefaultSession()`.
Retrieve a non-default BlackBerry Browser session.	> Invoke `Browser.getSession()`.

Request a web page.	> Invoke `BrowserSession.displayPage(String url)`, specifying the URL that contains the web content.
	`The following excerpt from the Restaurants.java sample creates a menu item that displays a web page in the BlackBerry Browser.`
	`private MenuItem browserItem = new MenuItem(_resources.getString(MENUITEM_BROWSER), 110, 12)`
	`{`
	`public void run() {`
	`synchronized(store) {String websiteUrl = websitefield.getText();`
	`if (websiteUrl.length == 0) {`
	`Dialog.alert(_resources.getString(ALERT_NO_WEBSITE));`
	`} else {`
	`BrowserSession visit = Browser.getDefaultSession();`
	`visit.displayPage(websiteUrl);`
	`}`
	`}`
	`}`
	`};`

Display content in a BlackBerry Browser field

To display web content in a BlackBerry® Browser field, use the `net.rim.blackberry.api.browser` package.

Task	Steps
Access a rendering session.	1. Invoke `RenderingSession.getNewInstance()`. 2. Store the returned rendering session handle in a `RenderingSession` object. `RenderingSession _renderingSession = RenderingSession.getNewInstance();.`
Define callback functionality for a rendering session.	> Implement the `RenderingApplication` interface.
Retrieve a BlackBerry Browser field.	1. Invoke `RenderingSession.getBrowserContent(javax.microedition.io.HttpConnection, net.rim.device.api.browser.field.RenderingApplication, net.rim.device.api.browser.field.Event)`. 2. Store the returned object in a `BrowserContent` object. You render web content in the `BrowserContent` object. `BrowserContent browserContent = _renderingSession.getBrowserContent(HttpConnection connection, this, Event e);`
Retrieve a field that renders the URL content to your application for display.	> Invoke `BrowserContent.getDisplayableContent()`, storing the returned object in a `Field` object. `Field field = browserContent.getDisplayableContent();`

Display a BlackBerry Browser field.	1. To clear the current screen, invoke the `MainScreen.deleteAll()` method.
	`_mainScreen.deleteAll();`
	2. To add field data to the application screen, invoke `MainScreen.add()`.
	`_mainScreen.add(field);`
	3. Create a non-main event thread to run `BrowserContent.finishLoading()` so that the UI does not lock.
	4. To render the new BlackBerry Browser content, invoke `BrowserContent.finishLoading()`. HTML files display a blank field until you invoke `BrowserContent.finishLoading()`. WML files and images might load before you invoke this method.

Create a separate thread for rendering.	> Create a non-main event thread that contains the instructions for retrieving and displaying the BlackBerry Browser field.
	`class CreationThread extends Thread {`
	`BrowserFieldHandlerApplication _callBackApplication;`
	`BasicRenderingApplication _renderingApplication;`
	`public CreationThread(BrowserFieldHandlerApplication callBackApplication) {`
	`_callBackApplication = callBackApplication;`
	`}`
	`public void run() {`
	`_renderingApplication = new BasicRenderingApplication(_callBackApplication);`
	`BrowserField field = _renderingApplication.getBrowserField("www.blackberry.com");`
	`_callBackApplication.displayBrowserField(field);`
	`}`
	`}`

Task	Steps
Set rendering options.	> Override `BrowserContent.getRenderingOptions()`. Your application uses the default rendering options if you do not override `BrowserContent.getRenderingOptions()`.

Manage events.	> Implement `RenderingApplication.eventOccurred()`, specifying the actions that occur when a specific rendering event occurs.
	The following example specifies actions that occur in the event of a URL request, change in browser content, or a redirect to a different web page.
	`public Object eventOccurred(Event event) {`
	`int eventId = event.getUID();`
	`switch (eventId) { case Event.EVENT_URL_REQUESTED : {`
	`UrlRequestedEvent urlRequestedEvent = (UrlRequestedEvent) event;`
	`String absoluteUrl = urlRequestedEvent.getURL();`
	`HttpConnection conn = null;`
	`PrimaryResourceFetchThread thread = new PrimaryResourceFetchThread(urlRequestedEvent.getURL(), urlRequestedEvent.getHeaders(), urlRequestedEvent.getPostData(),event, this);`
	`thread.start();`
	`break;}`
	`case Event.EVENT_BROWSER_CONTENT_CHANGED: {`
	`// The browser field title might have changed, so we update the title field.`
	`BrowserContentChangedEvent browserContentChangedEvent = (BrowserContentChangedEvent) event;`
	`if (browserContentChangedEvent.getSource() instanceof BrowserContent) {`
	`BrowserContent browserField = (BrowserContent) browserContentChangedEvent.getSource();`
	`String newTitle = browserField.getTitle();`
	`if (newTitle != null) {`
	`_mainScreen.setTitle(newTitle);}}`
	`break;`
	`}`
	`case Event.EVENT_REDIRECT : {`
	`RedirectEvent e = (RedirectEvent) event;`
	`String referrer = e.getSourceURL();`
	`switch (e.getType()) {`
	`case RedirectEvent.TYPE_JAVASCRIPT :`
	`break;`
	`case RedirectEvent.TYPE_META :`
	`// For MSIE and Mozilla, do not send a Referer for META Refresh.`
	`referrer = null;`
	`break;`
	`case Event.EVENT_SET_HEADER : // no cache support`
	`case Event.EVENT_SET_HTTP_COOKIE : // no cookie support`
	`default :`
	`}`
	`return null;`
	`}`

Code sample: Using the BlackBerry Browser

Example: BrowserFieldSampleApplication.java

```
/**
 * DefaultRenderingApplication.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.browser;

import java.io.IOException;
import javax.microedition.io.HttpConnection;
import net.rim.device.api.browser.field.*;
import net.rim.device.api.io.http.HttpHeaders;
import net.rim.device.api.system.Application;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.Status;
import net.rim.device.api.ui.container.MainScreen;


final public class BrowserFieldSampleApplication extends UiApplication implements
RenderingApplication
{
    private static final String REFERER = “referer”;
    private RenderingSession _renderingSession;
    private MainScreen _mainScreen;
    private HttpConnection  _currentConnection;

    public static void main(String[] args) {
        BrowserFieldSampleApplication app = new BrowserFieldSampleApplication();
        app.enterEventDispatcher();
    }

    private BrowserFieldSampleApplication() {
        _mainScreen = new MainScreen();
        pushScreen(_mainScreen);
        _renderingSession = RenderingSession.getNewInstance();

        PrimaryResourceFetchThread thread = new PrimaryResourceFetchThread(“http://
www.google.com”, null, null, null, this);
        thread.start();
    }

    public void processConnection(HttpConnection connection, Event e) {
        // cancel previous request
        if (_currentConnection != null) {
            try {
                _currentConnection.close();
            } catch (IOException e1) {
            }
        }
        _currentConnection = connection;
```

```
        BrowserContent browserContent = null;

        try {
            browserContent = _renderingSession.getBrowserContent(connection, this, e);

            if (browserContent != null) {
                Field field = browserContent.getDisplayableContent();
                if (field != null) {
                    synchronized (Application.getEventLock()) {
                        _mainScreen.deleteAll();
                        _mainScreen.add(field);
                    }
                }
                browserContent.finishLoading();
            }
        } catch (RenderingException re) {
        } finally {
            SecondaryResourceFetchThread.doneAddingImages();
        }
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#eventOccurred(net.rim.device.api.browser.E
vent)
     */
    public Object eventOccurred(Event event) {
        int eventId = event.getUID();
        switch (eventId) {
            case Event.EVENT_URL_REQUESTED : {
                UrlRequestedEvent urlRequestedEvent = (UrlRequestedEvent) event;
                String absoluteUrl = urlRequestedEvent.getURL();

                HttpConnection conn = null;
                PrimaryResourceFetchThread thread = new
PrimaryResourceFetchThread(urlRequestedEvent.getURL(),

urlRequestedEvent.getHeaders(),

urlRequestedEvent.getPostData(),
                                                                                event,
this);
                thread.start();
                break;
            } case Event.EVENT_BROWSER_CONTENT_CHANGED: {
                // browser field title might have changed update title
                BrowserContentChangedEvent browserContentChangedEvent =
(BrowserContentChangedEvent) event;
                if (browserContentChangedEvent.getSource() instanceof BrowserContent) {
                    BrowserContent browserField = (BrowserContent)
browserContentChangedEvent.getSource();
                    String newTitle = browserField.getTitle();
                    if (newTitle != null) {
                        _mainScreen.setTitle(newTitle);
                    }
                }
                break;
            } case Event.EVENT_REDIRECT : {
```

```
                    RedirectEvent e = (RedirectEvent) event;
                    String referrer = e.getSourceURL();
                    switch (e.getType()) {
                        case RedirectEvent.TYPE_SINGLE_FRAME_REDIRECT :
                            // show redirect message
                            Application.getApplication().invokeAndWait(new Runnable() {
                                public void run() {
                                    Status.show(“You are being redirected to a different
page...”);
                                }
                            });
                            break;
                        case RedirectEvent.TYPE_JAVASCRIPT :
                            break;
                        case RedirectEvent.TYPE_META :
                            // MSIE and Mozilla don’t send a Referer for META Refresh.
                            referrer = null;
                            break;
                        case RedirectEvent.TYPE_300_REDIRECT :
                            // MSIE, Mozilla, and Opera all send the original
                            // request’s Referer as the Referer for the new
                            // request.
                            Object eventSource = e.getSource();
                            if (eventSource instanceof HttpConnection) {
                                referrer =
((HttpConnection)eventSource).getRequestProperty(REFERER);
                            }
                            break;
                    }
                    HttpHeaders requestHeaders = new HttpHeaders();
                    requestHeaders.setProperty(REFERER, referrer);
                    PrimaryResourceFetchThread thread = new
PrimaryResourceFetchThread(e.getLocation(), requestHeaders,null, event, this);
                    thread.start();
                    break;
            } case Event.EVENT_CLOSE :
                // TODO: close the appication
                break;
            case Event.EVENT_SET_HEADER :        // no cache support
            case Event.EVENT_SET_HTTP_COOKIE :   // no cookie support
            case Event.EVENT_HISTORY :           // no history support
            case Event.EVENT_EXECUTING_SCRIPT :  // no progress bar is supported
            case Event.EVENT_FULL_WINDOW :       // no full window support
            case Event.EVENT_STOP :              // no stop loading support
            default :
        }

        return null;
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#getAvailableHeight(net.rim.device.api.brow
ser.BrowserContent)
     */
    public int getAvailableHeight(BrowserContent browserField) {
        // field has full screen
        return Graphics.getScreenHeight();
```

```
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#getAvailableWidth(net.rim.device.api.brows
er.BrowserContent)
     */
    public int getAvailableWidth(BrowserContent browserField) {
        // field has full screen
        return Graphics.getScreenWidth();
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#getHistoryPosition(net.rim.device.api.brow
ser.BrowserContent)
     */
    public int getHistoryPosition(BrowserContent browserField) {
        // no history support
        return 0;
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#getHTTPCookie(java.lang.String)
     */
    public String getHTTPCookie(String url) {
        // no cookie support
        return null;
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#getResource(net.rim.device.api.browser.Req
uestedResource,
     *      net.rim.device.api.browser.BrowserContent)
     */
   public HttpConnection getResource( RequestedResource resource, BrowserContent referrer)
{

        if (resource == null) {
            return null;
        }

        // Verify that this is a cache-only request.
        if (resource.isCacheOnly()) {
            // no cache support
            return null;
        }

        String url = resource.getUrl();

        if (url == null) {
            return null;
        }

        // If the referrer is null, return the connection.
        if (referrer == null) {
```

```
            HttpConnection connection = Utilities.makeConnection(resource.getUrl(),
resource.getRequestHeaders(), null);
            return connection;
        } else {
            // If the referrer is not null, set up the connection on a separate thread.
            SecondaryResourceFetchThread.enqueue(resource, referrer);
        }
        return null;
    }

    /**
     * @see
net.rim.device.api.browser.RenderingApplication#invokeRunnable(java.lang.Runnable)
     */
    public void invokeRunnable(Runnable runnable) {
        (new Thread(runnable)).run();
    }
}

class PrimaryResourceFetchThread extends Thread
{
    private BrowserFieldSampleApplication _application;
    private Event _event;
    private byte[] _postData;
    private HttpHeaders _requestHeaders;
    private String _url;

    PrimaryResourceFetchThread(String url, HttpHeaders requestHeaders, byte[] postData,
                               Event event, BrowserFieldSampleApplication application) {

        _url = url;
        _requestHeaders = requestHeaders;
        _postData = postData;
        _application = application;
        _event = event;
    }

    public void run() {
        HttpConnection connection = Utilities.makeConnection(_url, _requestHeaders,
_postData);
        _application.processConnection(connection, _event);
    }
}
```

14

Using location information

Types of location information
Using BlackBerry Maps
Locating BlackBerry devices using GPS information

Types of location information

Information Type	Description
Location Based Services	Location Based Services is a location-enabled platform that lets an application use location data on a BlackBerry® device or BlackBerry Enterprise Server.
GPS	The Location API (`javax.microedition.location`) lets applications retrieve the GPS location of the BlackBerry device. The GPS location is the geographical coordinates, latitude and longitude, of the BlackBerry device. Depending on the location method, applications might also retrieve the speed, orientation, and course of the BlackBerry device.

Using BlackBerry Maps

BlackBerry® Maps is a mapping and location client application that integrates into a subset of the existing BlackBerry applications. BlackBerry Maps lets you view a map for any location in the world, view and track the current location of the BlackBerry device, display a route from a start location to a specific destination, and search for and display points of interest on a map. You can invoke BlackBerry Maps from an application to display map information.

Location documents

Location documents store map information and are the protocol and container for interacting with the BlackBerry® Maps application. To display locations on a map, you must create location documents and fill in the data tags with location information.

Create a location document to display map data

Location document tags

Tag name	Tag number	Type	Description
lat	1	int	Latitude in degrees * 100000
lon	2	int	Longitude in degrees * 100000
zoom	3	int	Zoom
label	4	String	Name
description	5	String	Description

Location document structure

When creating a location document, use the following structure:

```
<location-document>
<location lat='<latitude>' lon='<longitude>' label='<Location_Label>'
description='<Description>'/>
<location lat='<latitude>' lon='<longitude>' label='<Location_Label>'
description='<Description>'/>
<location lat='<latitude>' lon='<longitude>' label='<Location_Label>'
description='<Description>'/>....</location-document>
```

Use the BlackBerry Maps application

Task	Steps
Open the default map view.	> Invoke `invokeApplication()` using a new `MapsArguments` object. `Invoke.invokeApplication( Invoke.APP_TYPE_MAPS, new MapsArguments() );`

Show a location in a map.	1. Create a location string that contains location data for a location.
	`String document = "<location-document><location lat='-8030000' lon='4326000' label='Kitchener, ON' description='You are here' zoom='10'/></location-document>";`
	2. Invoke `invokeApplication()` using the `APP_TYPE_MAPS` constant parameter, a new `MapsArguments` object that uses the ARG_LOCATION_DOCUMENT property and the location string.
	`Invoke.invokeApplication(Invoke.APP_TYPE_MAPS, new MapsArguments( MapsArguments.ARG_LOCATION_DOCUMENT, document));`

Show multiple locations on a map.	1. Create a location string that contains location data for multiple locations.
	`String document = "<location-document>" + "<location lat='-8030000' lon='4326000' label='Kitchener, ON'/>" + "<location lat='-7938000' lon='4341000' label='Toronto, ON'/>"`
	`+ "<location lat='-7540000' lon='4519000' label='Ottawa, ON'/>" + "</location-document>";`
	2. Invoke `invokeApplication()` using the APP_TYPE_MAPS constant parameter, a new `MapsArguments` object that uses the ARG_LOCATION_DOCUMENT property and the location string.
	`Invoke.invokeApplication(Invoke.APP_TYPE_MAPS, new MapsArguments( MapsArguments.ARG_LOCATION_DOCUMENT, document));`

Locating BlackBerry devices using GPS information

To let your application obtain the geographical coordinates, latitude and longitude, of a BlackBerry® device, use the Location API (javax.microedition.location). Depending on the location method you use, the application might also retrieve the speed, orientation, and course information of a BlackBerry device.

Methods for retrieving a GPS location

Method	Constant	Description
Cellsite	`GPS_AID_MODE_CELLSITE`	The cellsite method uses the GPS location of the active cellsite tower to provide first-order GPS information. **Advantages** • Is the fastest location method. **Disadvantages** • Provides low-level accuracy location information. It does not provide orientation, course, or speed information. **Requirements** • Requires support for network connectivity and wireless service providers.

Method	Constant	Description
Assisted	`GPS_AID_MODE_ASSIST`	The assisted method uses the wireless network to provide ephemeris GPS satellite data to the BlackBerry® device chip. **Advantages** • Provides the GPS location faster than the autonomous method and more accurately than the cellsite method. **Requirements** • Requires network connectivity and wireless service provider support.
Autonomous	`GPS_AID_MODE_AUTONOMOUS`	The autonomous method uses the GPS chip on the BlackBerry device without assistance from the wireless network. **Advantages** • Does not require assistance from the wireless network. **Disadvantages** • Is the slowest location method.

Selecting a GPS location provider

Specify the desired criteria by creating an instance of the `javax.microedition.location.Criteria` class, invoking the appropriate `set` methods, and then passing the instance to `LocationProvider.getInstance()`.

Note: To create a `LocationProvider` instance with default criteria, invoke `LocationProvider.getInstance(null)`.

The GPS location provider is selected depending on how closely the GPS location provider matches the defined criteria.

Specify criteria for selecting a GPS location provider

Task	Steps
Create selection criteria.	> Create an instance of a `Criteria` object. `Criteria criteria = new Criteria();`
Set whether BlackBerry® device users can incur cost.	> Invoke `setCostAllowed()`. `criteria.setCostAllowed(true);`
Set required horizontal accuracy.	> Invoke `setHorizontalAccuracy()`. `criteria.setHorizontalAccuracy(50);`
Set required vertical accuracy.	> Invoke `setVerticalAccuracy()`. `criteria.setVerticalAccuracy(50);`
Provide criteria to the GPS location provider.	> Invoke `LocationProvider.getInstance()`, storing the returned object in a `LocationProvider` object. `LocationProvider provider = LocationProvider.getInstance(criteria);`

Recommended GPS location method	Horizontal accuracy	Vertical accuracy	Cost	Power consumption
autonomous	required	required	not allowed	not applicable

Recommended GPS location method	Horizontal accuracy	Vertical accuracy	Cost	Power consumption
autonomous	required	required	allowed	low, medium, or no requirement
first fix: assisted subsequent fixes: autonomous	required	required	allowed	high
autonomous	not required	not required	not allowed	medium, high, or no requirement
assisted	not required	not required	allowed	medium or no requirement
first fix: assisted subsequent fixes: autonomous	not required	not required	allowed	high
cellsite	not required	not required	allowed	low

Note: GPS might become unavailable if the BlackBerry device has an obstructed view of GPS satellites, which can occur when the BlackBerry device is indoors or surrounded by buildings, trees, or dense clouds.

Retrieve BlackBerry device GPS location information

Task	Steps
Specify a response time for retrieving the location of the BlackBerry® device.	The time it takes to retrieve the location of the BlackBerry device for the first time depends on several factors, such as the selected GPS mode and the GPS signal strength. In autonomous method, typical times are less than 2 minutes. In assisted mode, typical times are less than 30 seconds. > Invoke `Criteria.setPreferredResponseTime()`, providing the desired response time in milliseconds.

Retrieve the location information for a BlackBerry device.	> In a non-event thread, invoke `LocationProvider.getLocation(int)`, providing a timeout in seconds.
	`try {`
	`// Specify -1 to have the implementation use its default timeout value`
	`// for this provider.`
	`Location location = provider.getLocation(-1);`
	`} catch (Exception e) {`
	`// handle LocationException, InterruptedException, SecurityException`
	`// and IllegalArgumentException`
	`}`

Retrieve the speed of a BlackBerry device.	1. Invoke `location.getQualifiedCoordinates`, storing the returned object in a `QualifiedCoordinates` object.
	`QualifiedCoordinates coordinates = location.getQualifiedCoordinates;`
	2. Invoke `location.getSpeed()`.
	`float speed = location.getSpeed();`

Retrieve the course of a BlackBerry device.	1. Invoke `location.getQualifiedCoordinates`, storing the returned object in a `QualifiedCoordinates` object.
	`QualifiedCoordinates coordinates = location.getQualifiedCoordinates;`
	2. Invoke `location.getCourse()`.
	`float course = location.getCourse();`

Task	Steps

Register a location listener.	You can associate only one location listener with a particular GPS location provider. Applications typically listen for updates on a separate thread.
	1. Implement the `LocationListener` interface.
	2. Register your implementation by invoking `LocationProvider.setLocationListener()`.
	`import javax.microedition.LocationProvider.*;`
	`public class SampleLocationApp {`
	`public static void main (string[] Args) {`
	`// ...`
	`provider.setLocationListener(new SampleLocationListener(), 0, 60, 60);`
	`}`
	`}`
	`class SampleLocationListener implements LocationListener {`
	`void locationUpdated(LocationProvider provider, Location location) {`
	`// Respond to the updated location.`
	`// If the application registered the location listener with an interval of`
	`// 0, the location provider does not provide location updates.`
	`}`
	`void providerStateChanged(LocationProvider provider, int newState) {`
	`switch (newState) {`
	`case LocationProvider.AVAILABLE :`
	`// The location provider is available.`
	`break;`
	`case LocationProvider.OUT_OF_SERVICE :`
	`// The location provider is permanently unavailable.`
	`// Consider cancelling the location listener by calling`
	`// provider.setLocationListener() with null as the listener.`
	`break;`
	`case LocationProvider.TEMPORARILY_UNAVAILABLE :`
	`// The location provider is temporarily unavailable.`
	`break;`
	`}`
	`}`
	`}`

Requirements for retrieving GPS location information from BlackBerry devices that run on the CDMA network

If you create applications for the BlackBerry® 7250 Wireless Handheld™ and the BlackBerry 7130e™ devices that require location information, you must obtain a unique PDE IP and port number from the CDMA wireless service provider for those BlackBerry devices. The PDE provides a BlackBerry device with initial ephemeris and almanac data when a BlackBerry device user starts an application that uses the Location API.

Before your application accesses the Location API, you must make sure that your application sends the PDE IP and port number to the wireless transceiver of the BlackBerry device. If an application on a CDMA device attempts to access the Location API without providing the correct PDE information to the wireless transceiver of the BlackBerry device, the application cannot obtain location information.

Send required PDE data to BlackBerry devices that run on the CDMA network

Before your application tries to use the Location API, send your PDE data to the wireless transceiver of the BlackBerry device by invoking the `setPDEInfo(String ip, int port)` method of the `GPSSettings` class (see the net.rim.device.api.gps package), using as parameters the PDE IP and port number that the CDMA wireless service provider provides to you.

Code sample: Recording GPS information for a BlackBerry device

Example: Subset of GPSDemo.java

```
/**
 * A GPS sample application using the JSR 179 APIs.
 *
 * Copyright (C) 2005 Research In Motion Limited.
 */

package com.rim.samples.docs.gpsdemo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;
import javax.microedition.location.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.resource.*;

/* This application acts as a simple travel computer, recording route coordinates,
 * speed, and altitude.
 * Recording begins as soon as the application is invoked.
 */
public class GPSDemo extends UiApplication implements GPSDemoResResource
{
    // Constants. ------------------------------------------------------------
    // The number of updates in seconds over which the altitude is calculated.
    private static final int GRADE_INTERVAL=5;
    // com.rim.samples.docs.gpsdemo.GPSDemo.ID
    private static final long ID = 0x4e94d9bc9c54fed3L;
    private static final int CAPTURE_INTERVAL=10;
    private static final int SENDING_INTERVAL=100;

    // Statics. --------------------------------------------------------------
    private static ResourceBundle _resources =
ResourceBundle.getBundle(GPSDemoResResource.BUNDLE_ID, GPSDemoResResource.BUNDLE_NAME);

    private static String _hostName = "<fill in the addres of the server here>:5555";
    private static int _interval = 1; //seconds - this is the period of position query
    private static Vector _previousPoints;
```

```
private static float[] _altitudes;
private static float[] _horizontalDistances;
private static PersistentObject _store;

// Initialize or reload the persistent store.
static {
    _store = PersistentStore.getPersistentObject(ID);
    if(_store.getContents()==null) {
        _previousPoints= new Vector();
        _store.setContents(_previousPoints);
    }
    _previousPoints=(Vector)_store.getContents();
}

private long _startTime;
private float _wayHorizontalDistance;
private float _horizontalDistance;
private float _verticalDistance;
private ListField _listField;
private EditField _status;
private StringBuffer _messageString;
private String _oldmessageString;
private LocationProvider _locationProvider;
private ServerConnectThread _serverConnectThread;

private final class GPSDemoScreen extends MainScreen
{
    protected void makeMenu(Menu menu, int instance)
    {
        menu.add( _markWayPoint );
        menu.add( _viewWayPoints );
        menu.add( _options );
        menu.add( _close );

        menu.addSeparator();

        super.makeMenu(menu, instance);
    }

    public void close() {
        if ( _locationProvider != null )
        {
            _locationProvider.reset();
            _locationProvider.setLocationListener(null, -1, -1, -1);
        }
        if ( _serverConnectThread != null )
        {
            _serverConnectThread.stop();
        }
        super.close();
    }
}

/* Instantiate the new application object and enter the event loop.
 * @param args unsupported. no args are supported for this application
 */
public static void main(String[] args) {
    new GPSDemo().enterEventDispatcher();
```

```
    }

// Constructors. -------------------------------------------------------------
    public GPSDemo() {
        // Used by waypoints; represents the time since the last waypoint.
        _startTime = System.currentTimeMillis();
        _altitudes=new float[GRADE_INTERVAL];
        _horizontalDistances=new float[GRADE_INTERVAL];
        _messageString= new StringBuffer();
        MainScreen screen = new GPSDemoScreen();
        screen.setTitle(new LabelField(_resources.getString(GPSDEMO_TITLE),
LabelField.USE_ALL_WIDTH));

        _status = new EditField();
        screen.add(_status);

        // try to start the GPS thread that listens for updates
        if ( startLocationUpdate() ) {
            startServerConnectionThread(); // if successful, start the thread that
communicates with the server
        }

        // Render our screen.
        pushScreen(screen);
    }

    /* Update the GUI with the data just received.
     */
    private void updateLocationScreen(final String msg) {
        invokeLater(new Runnable() {
            public void run() {
                _status.setText(msg);
            }
        });
    }
// Menu items. ---------------------------------------------------------------
    // Cache the markwaypoint menu item for reuse.
    private MenuItem _markWayPoint = new MenuItem(_resources,
GPSDEMO_MENUITEM_MARKWAYPOINT, 110, 10) {
        public void run() {
            GPSDemo.this.markPoint();
        }
    };

    // Cache the view waypoints menu item for reuse.
    private MenuItem _viewWayPoints = new MenuItem(_resources,
GPSDEMO_MENUITEM_VIEWWAYPOINTS, 110, 10) {
        public void run() {
            GPSDemo.this.viewPreviousPoints();
        }
    };

    // Cache the options menu item for reuse.
   private MenuItem _options = new MenuItem(_resources, GPSDEMO_MENUITEM_OPTIONS, 110, 10)
{
        public void run()
        {
            GPSDemo.this.viewOptions();
```

```
        }
    };

    // Cache the close menu item for reuse.
    private MenuItem _close = new MenuItem(_resources, GPSDEMO_MENUITEM_CLOSE, 110, 10) {
        public void run() {
            System.exit(0);
        }
    };

    /* Invokes the Location API with the default criteria.
     */
    private boolean startLocationUpdate() {
        boolean retval = false;

        try {
            _locationProvider = LocationProvider.getInstance(null);

            if ( _locationProvider == null ) {
                // We would like to display a dialog box indicating that GPS isn't
supported, but because
               // the event-dispatcher thread hasn't been started yet, modal screens cannot
be pushed onto
                // the display stack.  So delay this operation until the event-dispatcher
thread is running
                // by asking it to invoke the following Runnable object as soon as it can.
                Runnable showGpsUnsupportedDialog = new Runnable() {
                    public void run() {
                        Dialog.alert("GPS is not supported on this platform, exiting...");
                        System.exit( 1 );
                    }
                };
               invokeLater( showGpsUnsupportedDialog );  // ask event-dispatcher thread to
display dialog ASAP
            } else {
                // A single listener can be associated with a provider,
                // and unsetting it involves the same call but with null,
                // so there is no need to cache the listener instance.
                // Request an update every second.
                _locationProvider.setLocationListener(new LocationListenerImpl(),
_interval, 1, 1);
                retval = true;
            }
        } catch (LocationException le) {
            System.err.println("Failed to add a location listener. Exiting...");
            System.err.println(le);
            System.exit(0);
        }

        return retval;
    }
    /**
     * Invokes a separate thread used to send data to the server
     */
    private void startServerConnectionThread()
    {
        _serverConnectThread = new ServerConnectThread(_hostName);
        _serverConnectThread.start();
```

```
    }
    /* Marks a point in the persistent store. Calculations are based on
     * all data collected since the previous waypoint, or from the start
     * of the application if no previous waypoints exist.
     */
    private void markPoint() {
        long current = System.currentTimeMillis();
        WayPoint p= new WayPoint(_startTime, current, _wayHorizontalDistance,
_verticalDistance);

        addWayPoint(p);

        // Reset the waypoint variables.
        _startTime = current;
        _wayHorizontalDistance = 0;
        _verticalDistance = 0;
    }

    /**
     * View the various options for this application
     */
    public void viewOptions()
    {
        OptionScreen optionScreen = new OptionScreen();
        pushScreen(optionScreen);
    }

    // View the saved waypoints.
    private void viewPreviousPoints() {
        PointScreen pointScreen = new PointScreen(_previousPoints, _resources);
        pushScreen(pointScreen);
    }

    /* Adds a new waypoint and commits the set of saved waypoints
     * to flash memory.
     * @param p The point to add.
     */
    /*package*/ synchronized static void addWayPoint(WayPoint p) {
        _previousPoints.addElement(p);
        commit();
    }

    /* Removes a waypoint from the set of saved points and commits the modifed set to flash
memory.
     * @param p the point to remove
     */
    /*package*/ synchronized static void removeWayPoint(WayPoint p) {
        _previousPoints.removeElement(p);
        commit();
    }

    // Commit the waypoint set to flash memory.
    private static void commit() {
        _store.setContents(_previousPoints);
        _store.commit();
    }

    /**
```

```
     * Rounds off a given double to the provided number of decimal places
     * @param d the double to round off
     * @param decimal the number of decimal places to retain
     * @return a double with the number of decimal places specified
     */
    private static double round(double d, int decimal)
    {
        double powerOfTen = 1;
        while (decimal-- > 0)
        {
            powerOfTen *= 10.0;
        }
        double d1 = d * powerOfTen;
        int d1asint = (int)d1; //clip the decimal portion away and cache the cast, this is
a costly transformation
        double d2 = d1 - d1asint; //get the remainder of the double
        //is the remainder > 0.5? if so, round up, otherwise round down (lump in .5 with >
case for simplicity)
        return ( d2 >= 0.5 ? (d1asint + 1)/powerOfTen : (d1asint)/powerOfTen);
    }

    /**
     * Implementation of the LocationListener interface.
     */
    private class LocationListenerImpl implements LocationListener {
        // Members. ------------------------------------------------------------
        private int captureCount;
        private int sendCount;

        // Methods. ------------------------------------------------------------
        public void locationUpdated(LocationProvider provider, Location location) {
            if(location.isValid()) {
                float heading = location.getCourse();
                double longitude = location.getQualifiedCoordinates().getLongitude();
                double latitude = location.getQualifiedCoordinates().getLatitude();
                float altitude = location.getQualifiedCoordinates().getAltitude();
                float speed = location.getSpeed();

                // Horizontal distance.
                float horizontalDistance = speed * _interval;
                _horizontalDistance += horizontalDistance;
                // Horizontal distance for this waypoint.
                _wayHorizontalDistance += horizontalDistance;

                // Distance over the current interval.
                float totalDist = 0;
                // Moving average grade.
                for(int i = 0; i < GRADE_INTERVAL - 1; ++i) {
                    _altitudes[i] = _altitudes[i+1];
                    _horizontalDistances[i] = _horizontalDistances[i+1];
                    totalDist = totalDist + _horizontalDistances[i];
                }
                _altitudes[GRADE_INTERVAL-1] = altitude;
                _horizontalDistances[GRADE_INTERVAL-1] = speed*_interval;
                totalDist= totalDist + _horizontalDistances[GRADE_INTERVAL-1];
                float grade = (totalDist==0.0F)? Float.NaN : ( (_altitudes[4] -
_altitudes[0]) * 100/totalDist);
```

```
                // Running total of the vertical distance gain.
               float altGain = _altitudes[GRADE_INTERVAL-1] - _altitudes[GRADE_INTERVAL-2];
                if (altGain > 0) _verticalDistance = _verticalDistance + altGain;
                captureCount += _interval;

                // If we're mod zero then it's time to record this data.
                captureCount %= CAPTURE_INTERVAL;
                // Information to be sent to the server
                if ( captureCount == 0 )
                {
                   //minimize garbage creation by appending only character primitives, no
extra String objects created that way
                    _messageString.append(round(longitude,4));
                    _messageString.append(';');
                    _messageString.append(round(latitude,4));
                    _messageString.append(';');
                    _messageString.append(round(altitude,2));
                    _messageString.append(';');
                    _messageString.append(_horizontalDistance);
                    _messageString.append(';');
                    _messageString.append(round(speed,2));
                    _messageString.append(';');
                    _messageString.append(System.currentTimeMillis());
                    _messageString.append(':');
                    sendCount += CAPTURE_INTERVAL;
                    _horizontalDistance = 0;
                }

                //if we're mod zero then it's time to send
                sendCount %= SENDING_INTERVAL;
                synchronized(this)
                {
                    if (sendCount == 0 && _messageString.length() != 0)
                    {
                        if (_oldmessageString == null)
                        {
                            _oldmessageString = _messageString.toString();
                            //debug
                                System.out.println(_oldmessageString);
                        }
                        else
                        {
                            _oldmessageString = _oldmessageString +
_messageString.toString();
                            //debug
                                System.out.println(_oldmessageString);
                        }
                        _messageString.setLength(0);
                    _serverConnectThread.sendUpdate(_oldmessageString);
                    }
                }
                // Information to display on the device.
                StringBuffer sb = new StringBuffer();
                sb.append("Longitude: ");
                sb.append(longitude);
                sb.append("\n");
                sb.append("Latitude: ");
                sb.append(latitude);
```

```
            sb.append(“\n”);
            sb.append(“Altitude: “);
            sb.append(altitude);
            sb.append(“ m”);
            sb.append(“\n”);
            sb.append(“Heading relative to true north: “);
            sb.append(heading);
            sb.append(“\n”);
            sb.append(“Speed : “);
            sb.append(speed);
            sb.append(“ m/s”);
            sb.append(“\n”);
            sb.append(“Grade : “);
            if(Float.isNaN(grade))sb.append(“ Not available”);
            else sb.append(grade+” %”);
            GPSDemo.this.updateLocationScreen(sb.toString());
        }
    }

    public void providerStateChanged(LocationProvider provider, int newState) {
        // No operation defined.
    }
}
```

15

Creating push applications

Types of push applications
Types of push requests
Write a client push application
Write a server-side push application
Create a RIM push request
Create a PAP push request
Code sample: Pushing data to an application that listens on a BlackBerry device

Types of push applications

Push applications send web content or data to specific BlackBerry® device users. BlackBerry device users do not need to request or download the data because the push application delivers the information as soon as it becomes available.

Two types of push applications exist:

Application	Description
Browser push applications	Browser push applications send content to a web browser on the BlackBerry device. • The BlackBerry Browser configuration supports BlackBerry MDS™ Services push applications. • The WAP Browser configuration supports WAP push applications. • The Internet Browser configuration does not support push applications. See the *BlackBerry Browser Developer Guide* for more information about writing browser push applications.
Client/server push applications	A server-side application pushes data to a custom Java™ application on the BlackBerry device. Client/server push applications consist of a custom client application for the BlackBerry device and a server-side application that pushes content to the client application. This approach provides more control than browser push applications over the type of content that you can send and how the BlackBerry device processes and displays the content.

Types of push requests

Applications can send two types of push requests:

Request	Supported tasks	Push storage
RIM push	• sending a server-side push submission • specifying a reliability mode for the push submission • specifying a deliver-before time stamp for the push submission • requesting a result notification of the push submission • specifying a deliver-after time stamp for the push submission	RIM pushes are stored in RAM. Undelivered RIM pushes might be lost if the server reboots.
PAP	• sending a server-side push submission • specifying a reliability mode for the push submission • specifying a deliver-before time stamp for the push submission • requesting a result notification of the push submission • specifying a deliver-after time stamp for the push submission • cancelling a push request submission • querying the status of a push request submission **Note**: This request type is part of the WAP 2.0 specification. Visit http://www.openmobilealliance.org for more information about PAP.	PAP pushes are stored in a database.

Note: The BlackBerry® Mobile Data System™ queues only 1000 push requests, including both RIM and PAP push requests.

Write a client push application

Task	Steps
Create a listening thread.	> Send and receive data on a separate thread so that you do not block the main event.

Determine if a BlackBerry® device is in a wireless coverage area.	1. Create code to check if the IPPP service book can be routed.
	`if( ServiceBook.getSB().getRecordByUidAndCid( serviceUID, "IPPP" ) == null ) {`
	`// There is no service book`
	`return false;`
	`}`
	`if( ServiceRouting.getInstance().isServiceRoutable( serviceUID, -1 ) ) {`
	`// Serial bypass is active`
	`return true;`
	`}`
	2. Create code to check that the wireless transceiver is on and that data coverage is on.
	`return RadioInfo.getState() != RadioInfo.STATE_OFF && ( RadioInfo.getNetworkService() & RadioInfo.NETWORK_SERVICE_DATA ) != 0;`

Open an input stream.	1. Invoke `Connector.open(String)`, specifying **http://** as the protocol and choosing a high port number from 1 to 65,535 to avoid conflicts with other applications. You cannot use the following port numbers:
	• 80
	• 443
	• 7874
	• 8080
	`Connnector.open(http://6234);`
	2. To specify the connection type that the application uses for incoming and outgoing connections, at the end of the connection string, add a colon, followed by the optional `deviceside=<boolean>` parameter with one of the following values:
	• If the application listens for push information from BlackBerry MDS™ Services, set the deviceside=<*boolean*> parameter to **false**.
	• If the application listens for push information from WAP push requests, do not use the deviceside=<*boolean*> parameter.
	3. Cast the object that `Connector.open` returns as a `StreamConnectionNotifier`.
	`StreamConnectionNotifier _notify = (StreamConnectionNotifier)Connector.open("http://:6234");`
	4. Open a server-side stream connection once and keep the server-side stream connection open.
	`// open a server-side stream connection`
	`StreamConnection stream = _notify.acceptAndOpen();`
	`// open an input stream for the connection`
	`InputStream input = stream.openInputStream();`
	5. Read the incoming data.
	6. If you use application-level push reliability, use the `pushInputStream.accept()` method to accept and acknowledge the incoming data.
	7. If an `IOException` occurs, reopen the connection.

Task	Steps
Close the stream connection notifier.	> Invoke `close()` on the stream connection notifier. `_notify.close();`

Code sample: Listening for data from a web server

Example: HTTPPushDemo.java

```
/**
 * The client side of a simple HTTP Push system.
 * This application will listen for image data on the specified port and
 * render the data when it arrives.
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httppush;

import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;
import net.rim.device.api.util.*;
import net.rim.device.api.io.http.*;

public class HTTPPushDemo extends UiApplication implements HTTPPushDemoResResource
{
    // Constants.
    private static final String URL = “http://:100”; //PORT 100
    private static final int CHUNK_SIZE = 256;

    // Fields.
    private ListeningThread _listeningThread;
    private MainScreen _mainScreen;
    private RichTextField _infoField;
    private RichTextField _imageField;

    //statics ------------------------------------------------------------------
    private static ResourceBundle _resources =
ResourceBundle.getBundle(HTTPPushDemoResResource.BUNDLE_ID,
HTTPPushDemoResResource.BUNDLE_NAME);

    public static void main(String[] args) {
        HTTPPushDemo theApp = new HTTPPushDemo();
        theApp.enterEventDispatcher();
    }

    /**
    * Create a separate listening thread so that you do not
    * block the application’s main event thread.
    */
    private class ListeningThread extends Thread {
```

```
        private boolean _stop = false;
        private StreamConnectionNotifier _notify;

        public synchronized void stop() {
            _stop = true;
            try {
                _notify.close(); // Close the connection so thread returns.
            } catch (IOException e) {
                System.err.println(e.toString());
            } catch (NullPointerException e) {
                // The notify object likely failed to open, due to an IOException.
            }
        }


    public void run()
    {
    StreamConnection stream = null;
    InputStream input = null;
    MDSPushInputStream pushInputStream=null;
    while (!_stop)
     {
     try
      {
      //synchronize here so that we don't end up creating a
      //connection that is never closed
      synchronized(this)
      {
      // Open the connection once (or re-open after an IOException),
      // so we don't end up in a race condition, where a push is lost if
      // it comes in before the connection is open again.
      // we open the url with a parameter that indicates that we should always
      // use MDS when attempting to connect.
    _notify = (StreamConnectionNotifier)Connector.open(URL + ";deviceside=false");
            }

                    while (!_stop)
                    {
                        //NOTE: the following will block until data is received
                        stream = _notify.acceptAndOpen();
                        try {
                            input = stream.openInputStream();
                            pushInputStream= new
MDSPushInputStream((HttpServerConnection)stream, input);

                            //Extract the data from the input stream

                            DataBuffer db = new DataBuffer();
                            byte[] data = new byte[CHUNK_SIZE];
                            int chunk = 0;
                            while ( -1 != (chunk = input.read(data)) )
                            {
                                db.write(data, 0, chunk);
                            }

                            updateMessage(data);
                            //This method is called to accept the push
                            pushInputStream.accept();
```

```
                        input.close();
                        stream.close();

                        data = db.getArray();


                    } catch (IOException e1) {
                        // a problem occurred with the input stream
                       // however, the original StreamConnectionNotifier is still valid
                        System.err.println(e1.toString());

                        if ( input != null ) {
                            try {
                                input.close();
                            } catch (IOException e2) {
                            }
                        }
                        if ( stream != null ) {
                            try {
                                stream.close();
                            } catch (IOException e2) {
                            }
                        }
                    }
                }

                _notify.close();
                _notify = null;
            } catch (IOException ioe)
            {
                // likely the stream was closed
                System.err.println(ioe.toString());

                if ( _notify != null ) {
                    try {
                        _notify.close();
                        _notify = null;
                    } catch ( IOException e ) {
                    }
                }
            }
        }
    }
}

private final class HTTPMainScreen extends MainScreen
{
    public void close() {
        // Stop the listening thread.
        _listeningThread.stop();
        try {
            _listeningThread.join();
        } catch (InterruptedException e) {
            System.err.println(e.toString());
        }
        super.close();
```

```
        }
    }

    // Constructor.
    public HTTPPushDemo() {
        _mainScreen = new HTTPMainScreen();
        _mainScreen.setTitle(new LabelField(_resources.getString(HTTPPUSHDEMO_TITLE),
LabelField.USE_ALL_WIDTH));
        _infoField = new RichTextField();
        _mainScreen.add(_infoField);
        _mainScreen.add(new SeparatorField());
        _imageField = new RichTextField();
        _mainScreen.add(_imageField);

        _listeningThread = new ListeningThread();
        _listeningThread.start();

        _infoField.setText(_resources.getString(HTTPPUSHDEMO_LISTENTHREADSTARTED));
        pushScreen(_mainScreen);
    }

    private void updateMessage(final byte[] data)  {
        Application.getApplication().invokeLater(new Runnable() {
        public void run() {

        //query the user to load the received message
        String[] choices = {_resources.getString(HTTPPUSHDEMO_DIALOG_OK),
_resources.getString  (HTTPPUSHDEMO_DIALOG_CANCEL)};
       if ( 0 != Dialog.ask(_resources.getString(HTTPPUSHDEMO_QUERYFORRENDER), choices, 0)
)
        {
         return;
         }
        _infoField.setText(_resources.getString(HTTPPUSHDEMO_IMAGEINFO) + data.length);

         try {
           _imageField.setText(new String(data));
          }
         catch (Exception e) {
            Dialog.inform(e.toString());
            System.err.println(e.toString());
                }
            }
        });
    }
}
```

Write a server-side push application

To create a push application, you can use any programming language that can establish an HTTP connection . The following sections use standard Java™ to demonstrate a server-side push application.

Task	Steps
Specify a port.	If you create a client/server push application, you must make sure that the server-side application uses a port number other than 80, 443, 7874, and 8080 to deliver push data. > To specify a different port number, in the application, include the `X-Rim-Push-Dest-Port` header with the port value.
Connect to the BlackBerry® MDS™ Connection Service.	> Establish a connection using the fully qualified computer name or IP address.

Construct the push URL.	> To create a push request, perform one of the following actions:
	• Create a RIM push request using the following format:
	`/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=<uri>`
	`<headers>`
	`<content>`
	• Create a PAP push request using the following format:
	`/pap`
	See "Create a RIM push request" on page 225 for more information about RIM push requests.
	See "Create a PAP push request" on page 226 for more information about PAP push requests.

Connect to the BlackBerry Enterprise Server.	1. Invoke `openConnection()` on the push URL. 2. Cast the object that `url.openConnection()` returns as an `HttpURLConnection`. An `HttpURLConnection` represents a connection to a remote object. `HttpURLConnection conn =(HttpURLConnection)url.openConnection();`

Set properties for the HTTP POST request.	1. Create a `POST` request.
	`conn.setRequestMethod("POST"); // Post to the BlackBerry Enterprise Server.`
	2. To receive confirmation, set the parameter in `setDoInput(Boolean)` to `true` to indicate that the application intends to read data from the URL connection.
	`conn.setDoInput(true);`
	3. To send data, set the parameter in `setDoOutput(Boolean)` to `true` to indicate that the application intends to send data to the URL connection.
	`conn.setDoOutput(true);`

Write data to the server connection.	1. To access an output stream, invoke `getOutputStream()`.
	`OutputStream out = conn.getOutputStream();`
	2. Write data to the output stream.
	`out.write(data);`
	3. Close the output stream.
	`out.close();`

Task	Steps

Read the server response.	1. To access an input stream, invoke `getInputStream()`.
	`InputStream ins = conn.getInputStream();`
	2. Determine the size of the content. If the size of the content is non zero, open a data input stream, and then retrieve the content.
	`int contentLength = conn.getContentLength();`
	`if (contentLength > 0) {`
	`byte[] someArray = new byte [contentLength];`
	`DataInputStream dins = new DataInputStream(ins);`
	`dins.readFully(someArray);`
	`System.out.println(new String(someArray));`
	`}`
	`ins.close();`

Close the server connection.	> To indicate that the application will make no further requests to the server, invoke `disconnect()` `conn.disconnect();`

Work with a server-side push request

Send a request to cancel a PAP push submission.	1. Use the `cancel-message push-id` header. For example:
	`<cancel-message push-id="123@wapforum.org">`
	2. To specify the address to which the application submitted the push message, use the `address address-value` header. This is a required tag.
	`<address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net" />`
	The following example shows a PAP push cancellation request:
	`Content-Type: application/xml`
	`<?xml version="1.0"?>`
	`<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"`
	`"http://www.wapforum.org/DTD/pap_2.0.dtd">`
	`<pap>`
	`<cancel-message push-id="a_push_id">`
	`<address address-value=`
	`"WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/>`
	`</cancel-message>`
	`</pap>`
	Note: When pushing to a Group address, you cannot determine the status of delivery to a particular recipient or cancel delivery to one or more recipients. If the application requires this functionality, specify multiple recipient addresses in the push submission.

Query the status of a PAP push request.	1. To specify the push message on which you want to obtain status information, send a PAP push query request using the `statusquery-message push-id` header. For example:
	`<statusquery-message push-id="123@wapforum.org">`
	2. To specify the address to which you want to submit the push message, in the PAP push query request, use the `address-value` header. For example:
	`<address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net" />`
	The following example shows a RIM network status query request:
	`Content-Type: application/xml`
	`<?xml version="1.0"?>`
	`<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"`
	`"http://www.wapforum.org/DTD/pap_2.0.dtd"`
	`[<?wap-pap-ver supported-versions="2.0,1.*"?>]> <pap>`
	`<statusquery-message push-id="123@wapforum.org ">`
	`<address`
	`address-value="aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/>`
	`</statusquery-message>`
	`</pap>`

Create a RIM push request

Task	Steps

Push content to one or multiple BlackBerry® device users using a RIM push request.	1. To push data to a single BlackBerry device user using RIM push, send an HTTP POST request using the following URL format:
	`http://<mdsServer>:<web server listen port>/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=<uri><headers><content>`
	where:
	• <*destination*> is the destination PIN or email address
	• <*port*> is the destination port number
	• <*uri*> is the URI sent to the BlackBerry device
	• <*headers*> consists of HTTP headers
	• <*content*> is a byte stream
	2. To push content to multiple recipients using RIM push, include multiple DESTINATION parameters in the query string.
	`http://mds_server:8080/push?DESTINATION=user1@rim.com&DESTINATION=user2@rim.com&PORT=7874&REQUESTURI=/`

Push content to a group using RIM push.	> In the recipient addresses portion of the push submission, prefix the group name with the $ character. In the following example, the $ character is URL encoded. The following example shows a URL used to push to a group named IT using RIM push: `http://mds_server:8080/push?DESTINATION=%24IT&PORT=7874&REQUESTURI=/`
Specify a unique message ID to cancel or check the status of a message.	> Use the `X-RIM-Push-ID` header. Typically, specify a URL in combination with a value, such as 123@blackberry.com. If this header is omitted, the BlackBerry Mobile Data System™ generates a unique message ID. **Note:** Push identifiers must not end in @ppg.rim.com.
Specify a URL to send a result notification.	> Use the `X-RIM-Push-NotifyURL` header. The result notification contains the `X-RIM-Push-ID` header, which specifies the message ID, and the `X-RIM-Push-Status` header, which specifies an HTTP response code. The notification also contains an `X-RIM-Push-Destination` header that specifies the recipient address to which the result pertains.
Specify the delivery reliability mode of the content.	> Use the `X-RIM-Push-Reliability-Mode` header with one of the following modes: • application-level (`APPLICATION`) • application- preferred (`APPLICATION-PREFERRED`) • transport-level (`TRANSPORT`)
Specify the date and time by which to deliver the content to the BlackBerry device.	> Use the `X-RIM-Push-Deliver-Before-timestamp` header. Content that the application does not deliver before this date is not delivered. For example: `Mon, 28 Aug 2006 16:06:00 GMT`
Specify the date and time after which the content is delivered to the BlackBerry device. The application does not deliver the content before this date. Represent the date and time in UTC format.	> In the `X-RIM-Push-Deliver-After-timestamp` header, specify the date and time in UTC format. For example: `Mon, 28 Aug 2006 16:06:00 GMT`

Create a PAP push request

See "Appendix: XML control entity attributes" on page 275 for more information about XML control entity attributes.

Task	Steps

Push content to one or multiple BlackBerry® device users using PAP.	1. To push data to a single BlackBerry device user using PAP, send an `HTTP POST` request using the following format:
	`http://<mdsServer>:<web server listen port>/pap - The URL to send the PAP push to.`
	The request is a MIME multipart message, which consists of the following items:
	• XML document specifying the control entity
	• push content
	The following example shows a PAP push request:
	`Content-Type: multipart/related; type="application/xml"; boundary=asdlfkjiurwghasf`
	`X-Wap-Application-Id: /`
	`--asdlfkjiurwghasf`
	`Content-Type: application/xml`
	`<?xml version="1.0"?>`
	`<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"  "http://www.wapforum.org/DTD/pap_2.0.dtd">`
	`<pap>`
	`<push-message push-id="a_push_id" ppg-notify-requested-to="http://foo.rim.net/ReceiveNotify">`
	`<address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/>`
	`<quality-of-service delivery-method="unconfirmed"/>`
	`</push-message>`
	`</pap>`
	`--asdlfkjiurwghasf`
	`Content-Type: text/html`
	`<html><body>Hello, PAP world!</body></html>`
	`--asdlfkjiurwghasf--`
	2. To push content to multiple recipients using PAP, add multiple address tags to the post request. For example:
	`<address address-value="WAPPUSH=user1%40rim%2ecom%5B%3A7874/TYPE=USER@rim.net"></address>`
	`<address address-value="WAPPUSH=user2%40rim%2ecom%5D%3A7874/TYPE=USER@rim.net"></address>`

Push content to a group using PAP.	> In the recipient addresses part of the push submission, prefix the group name with the $ character. In the following example, the $ character is URL encoded. An address element is used to push to a group named IT using PAP: `<address address-value="WAPPUSH=%24IT/TYPE=USER@rim.net"/>`

Code sample: Pushing data to an application that listens on a BlackBerry device

The HTTPPush.java sample application, which uses standard Java™, sends a string of text to a listening client application on the BlackBerry® device using either a RIM push or a PAP push. The application pushes data based on an email address. To test push applications with the BlackBerry device simulator, define a mapping between the email address and the BlackBerry device simulator PIN (2100000A).

The following code sample compiles using J2SE 1.4.2.

Example: HTTPPushServer.java

```
/*
 * HttpPushServer.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httppush;

import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
import java.net.*;
import java.util.*;
import java.awt.Color;

/**
 * <p>The HTTPPushServer class provides a simple PUSH server sample.
 * <p>This program will send text to a listening device. The associated client demo
 * is HTTPPushServer. Start up both the device simulator and MDS before executing
 * this program. For reliable push, append the port that you are pushing to in
 * rimpublic.property file (push.application.reliable.ports):
 * <code>push.application.reliable.ports=7874,<b>100</b></code
 *
 * <p> The general form of the URL for posting (pushing) data to the device is:
 * http://&lt;host&gt;:&lt;port&gt;/push?DESTINATION=&lt;device
pin&gt;&amp;PORT=&lt;device_port&gt;&REQUESTURI=&lt;post uri&gt;
 */
public class HTTPPushServer extends javax.swing.JFrame {

    //constants ----------------------------------------------------------------
    private static final String RESOURCE_PATH = “com/rim/samples/docs/httppush/resources”;
    private static final String DEVICE_PIN = “2100000A”;
    private static final String DEVICE_PORT = “100”;
    private static final int MDS_PORT = 8080;

    private static final String IMAGE_TYPE = “.png”;
    private String requestTemplate;
    private String notifyURL=”http://localhost:7778”;
    private Random random= new Random();
    private Thread notificationThread;

    //statics ------------------------------------------------------------------
    private static ResourceBundle _resources =
java.util.ResourceBundle.getBundle(RESOURCE_PATH);
```

```
    //constructors ----------------------------------------------------------------
    /** Creates a new HTTPPushServer instance*/
    public HTTPPushServer() {

        initComponents ();
        pack ();

        //sizing code for the main frame
        setSize(_panel.getWidth(), _panel.getHeight());
        setLocation(100,100);
        notificationThread= new NotificationThread();
    }

    private URL getPushURL(String DevicePin)
    {
        /**
         * The format of the URL is:
         * http://<host>:<port>/push?DESTINATION=<device
pin>&PORT=<device_port>&REQUESTURI=<post uri>
         */
        URL _pushURL = null;
        try {
            if ((DevicePin == null) || (DevicePin.length() == 0))
            {
                DevicePin = DEVICE_PIN;
            }
            _pushURL = new URL("http", "localhost", MDS_PORT, "/push?DESTINATION="+
DevicePin +"&PORT="+DEVICE_PORT+"&REQUESTURI=localhost");

        } catch (MalformedURLException e) {
            System.err.println(e.toString());
        }
        return _pushURL;
    }

    /**
     * This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the FormEditor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        _panel = new javax.swing.JPanel();
        _textField = new javax.swing.JTextField();
        _textArea= new javax.swing.JTextArea();
        _pinField = new javax.swing.JTextField(DEVICE_PIN);
        _label = new javax.swing.JTextArea();
        _notification=new javax.swing.JTextArea();
        _rimButton= new javax.swing.JRadioButton("rim");
        _papButton= new javax.swing.JRadioButton("pap");
        _buttonGroup= new javax.swing.ButtonGroup();
        _buttonGroup.add(_rimButton);
        _buttonGroup.add(_papButton);

        _sendButton = new javax.swing.JButton();
        getContentPane().setLayout(null);
```

```
        setTitle(java.util.ResourceBundle.getBundle("com/rim/samples/docs/httppush/
resources").getString("HTTPPushServer.title"));
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        _panel.setLayout(null);
        _panel.setPreferredSize(getSize());

        _textArea.setToolTipText(java.util.ResourceBundle.getBundle("com/rim/samples/docs/
httppush/resources").getString("HTTPPushServer._textField.toolTipText"));
        _panel.add(_textArea);
        _textArea.setBounds(10, 50, 270, 100);
        _textArea.setBorder(new LineBorder(Color.BLACK));


        _pinField.setToolTipText(java.util.ResourceBundle.getBundle("com/rim/samples/docs/
httppush/resources").getString("HTTPPushServer._pinField.toolTipText"));
        _panel.add(_pinField);
        _pinField.setBounds(10, 170, 150, 30);

        _panel.add(_rimButton);
        _panel.add(_papButton);
        _rimButton.setBounds(170, 170, 50, 30);
        _papButton.setBounds(240, 170, 50, 30);

        _label.setWrapStyleWord(true);
        _label.setLineWrap(true);
        _label.setEditable(false);
        _label.setText(java.util.ResourceBundle.getBundle("com/rim/samples/docs/httppush/
resources").getString("HTTPPushServer._label.text"));
        _label.setBackground((java.awt.Color) javax.swing.UIManager.getDefaults ().get
("Button.background"));
        _panel.add(_label);
        _label.setBounds(10, 10, 270, 40);

        _sendButton.setLabel(java.util.ResourceBundle.getBundle("com/rim/samples/docs/
httppush/resources").getString("HTTPPushServer._sendButton.label"));
        _sendButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                sendButtonMouseClicked(evt);
            }
        });

        _panel.add(_sendButton);
        _sendButton.setLocation(10, 210);
        _sendButton.setSize(_sendButton.getPreferredSize());

        JScrollPane _scrollPane = new javax.swing.JScrollPane(_notification);
        _scrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        _panel.add(_scrollPane);
        _scrollPane.setBounds(10,250,270, 150);
```

```
        getContentPane().add(_panel);
        _panel.setBounds(0, 0, 300, 450);

    }//GEN-END:initComponents

    private void sendButtonMouseClicked(java.awt.event.MouseEvent evt)
    {//GEN-FIRST:event_sendButtonMouseClicked

        String text =_textArea.getText();

        if(_rimButton.isSelected()) postData(text);
        else if(_papButton.isSelected()) papPush(text);

    }//GEN-LAST:event_sendButtonMouseClicked

   /**
    * <p>posts the specified data to the device
    * <p>The URL is hardcoded for the purposes of this demo, and takes the form:
    * http://&lt;host&gt;:&lt;port&gt;/push?DESTINATION=&ltdevice
pin&gt;&PORT=&lt;device_port&gt;&REQUESTURI=&lt;post uri&gt;
    * param data the data to post
    *
    */
    private void postData(String data)
    {

        String pushId="pushID:"+random.nextInt();

        setupNotifyThread();

        try {
            URL url = getPushURL(_pinField.getText());

System.out.println(_resources.getString("HTTPPushServer.status.sendingToString") +
url.toString());

            //open the connection using the static member...
            HttpURLConnection conn =(HttpURLConnection)url.openConnection();
            conn.setDoInput(true);//For receiving the confirmation
            conn.setDoOutput(true);//For sending the data
            conn.setRequestMethod("POST");//Post the data to the proxy
            conn.setRequestProperty("X-RIM-PUSH-ID", pushId);
            conn.setRequestProperty("X-RIM-Push-NotifyURL", notifyURL);
            conn.setRequestProperty("X-RIM-Push-Reliability-Mode","APPLICATION");
            //Write the data
            OutputStream out = conn.getOutputStream();
            out.write(data.getBytes());
            out.close();
            InputStream ins =conn.getInputStream();
            int contentLength =conn.getContentLength();
            System.out.println(
_resources.getString("HTTPPushServer.status.contentLengthDescription")+ contentLength);
            if (contentLength > 0)
            {
                byte[] someArray = new byte [contentLength];
                DataInputStream dins = new DataInputStream(ins);
                dins.readFully(someArray);
```

```
                System.out.println(new String(someArray));
            }

            conn.disconnect();

        } catch (IOException e) {
            System.err.println(e);
        }
    }



    private void readPapTemplate()
    {

        try {
            String papFilename =  “com/rim/samples/docs/httppush/pap_push.txt”;
            InputStream ins = new BufferedInputStream(new FileInputStream(papFilename));
            ByteArrayOutputStream bouts = new ByteArrayOutputStream();
            copyStreams(ins, bouts);
            this.requestTemplate = new String(bouts.toByteArray());
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    private void setupNotifyThread()
    {
        if( !notificationThread.isAlive() )
        {
            notificationThread = new NotificationThread();
            notificationThread.start();
        }
    }


    private void papPush(String data)
    {
        String pushId=”pushID:”+random.nextInt();

        setupNotifyThread();

        readPapTemplate();
        String errorCode = null;
        try {
            String mdsHost = “localhost”;
            URL mdsUrl = new URL(“http”, mdsHost, MDS_PORT, “/pap”);
           System.out.println(“ sending PAP request to “ + mdsUrl.toString() + “; pushId =
“ + pushId);

            HttpURLConnection mdsConn = (HttpURLConnection)mdsUrl.openConnection();

            String boundary = ““;
            boundary = “asdlfkjiurwghasf”;
            mdsConn.setRequestProperty(“Content-Type”, “multipart/related;
type=\”application/xml\”; boundary=” + boundary);
            mdsConn.setRequestProperty(“X-Wap-Application-Id”, “/”);
            mdsConn.setRequestProperty(“X-Rim-Push-Dest-Port”,”100”);
```

```
            mdsConn.setRequestMethod("POST");

            mdsConn.setAllowUserInteraction(false);
            mdsConn.setDoInput(true);
            mdsConn.setDoOutput(true);

            String output = requestTemplate.replaceAll("\\$\\(pushid\\)", pushId);
            output = output.replaceAll("\\$\\(boundary\\)", boundary);
            output = output.replaceAll("\\$\\(notifyURL\\)", "" + notifyURL);
            output = output.replaceAll("\\$\\(pin\\)", "" + _pinField.getText());

            String   deliveryMethod = "confirmed";

            output = output.replaceAll("\\$\\(deliveryMethod\\)", deliveryMethod);


            output = output.replaceAll("\\$\\(headers\\)", "Content-Type: text/plain");
            output = output.replaceAll("\\$\\(content\\)", data);

            output = output.replaceAll("\r\n", "EOL");
            output = output.replaceAll("\n", "EOL");
            output = output.replaceAll("EOL", "\r\n");


            System.out.println(output);
            OutputStream outs = mdsConn.getOutputStream();
            copyStreams(new ByteArrayInputStream(output.getBytes()), outs);

            mdsConn.connect();

            ByteArrayOutputStream response = new ByteArrayOutputStream();
            copyStreams(mdsConn.getInputStream(), response);

            int httpCode = mdsConn.getResponseCode();

            if (httpCode != HttpURLConnection.HTTP_ACCEPTED) {
                throw new Exception("MDS returned HTTP status: " + httpCode);
            }

        } catch (Exception exception) {
            if (errorCode == null)
            {
                errorCode = exception.getClass().getName();
            }

            System.out.println(" encountered error on submission: " +
exception.toString());
        }
    }

    public void copyStreams(InputStream ins, OutputStream outs) throws IOException
    {
            int maxRead = 1024;
            byte [] buffer = new byte[1024];
            int bytesRead;

            for(;;)
            {
```

```
                bytesRead = ins.read(buffer);
                System.out.println(buffer);
                if (bytesRead <= 0) break;
                outs.write(buffer, 0, bytesRead);
            }
        }

        /** Exit the Application */
        private void exitForm(java.awt.event.WindowEvent evt) {//GEN-FIRST:event_exitForm
            System.exit (0);
        }//GEN-LAST:event_exitForm

        /**
        * @param args the command line arguments
        */
        public static void main (String args[]) {
            new HTTPPushServer().show ();
        }

        // Variables declaration - do not modify//GEN-BEGIN:variables
        private javax.swing.JPanel _panel;
        private javax.swing.JTextField _textField;
        private javax.swing.JTextArea _textArea;
        private javax.swing.JTextField _pinField;
        private javax.swing.JTextArea _label;
        private javax.swing.JTextArea _notification;
        private javax.swing.JButton _sendButton;
        private javax.swing.JRadioButton _rimButton;
        private javax.swing.JRadioButton _papButton;
        private javax.swing.ButtonGroup _buttonGroup;
        private javax.swing.JScrollPane _scrollPane;

        // End of variables declaration//GEN-END:variables

        public class NotificationThread extends Thread {

            public void run()
            {
                try {
                    System.out.println("Waiting for notification on port " + 7778 + "...");
                    while (true)
                    {
                        ServerSocket serverSocket = new ServerSocket(7778);
                        serverSocket.setSoTimeout(120000);
                        try {
                            Socket clientSocket = serverSocket.accept();
                            _notification.setText("Received notification:");
                            InputStream input = clientSocket.getInputStream();
                            StringBuffer str= new StringBuffer();
                            int byteRead = input.read();
                            while ((byteRead != -1) && (input.available() > 0))
                            {
                                str.append((char)byteRead);
                                byteRead = input.read();
                            }
                            _notification.append(str.toString());
                            PrintWriter output = new
PrintWriter(clientSocket.getOutputStream());
```

```
                        output.close();
                        clientSocket.close();
                    } catch (SocketTimeoutException ste) {
                        System.out.println("Notification connection timeout.
Restarting...");
                    }
                    serverSocket.close();
                }
            } catch (Exception exception) {
                exception.printStackTrace();
            }

        }
    }

}
```

16

Localizing applications

Storing text strings in resource files
Storing resources for a locale
Files required for localization
Add localization support
Code sample: Storing text strings in separate resources for locales
Retrieve strings from a resource file
Code sample: Retrieving strings from a resource file

Storing text strings in resource files

Design applications so that they are localized (adapted to specific languages and regions) without coding changes. Instead of including textual elements in your source code, store text strings in separate resource files. In your source code, use unique identifiers that map to the appropriate resource files.

Storing text strings in separate resource files has two benefits:

- Text translation is efficient because all of the text strings for a given locale are stored in a single file, outside your source code.
- Applications can dynamically retrieve the appropriate text to display to the BlackBerry® device user, based on the locale of the BlackBerry device user.

The BlackBerry Java™ Development Environment includes a mechanism for creating string resources. The `net.rim.device.api.i18n` package includes the Localization API.

Note: MIDP applications do not support localization.

A `ResourceBundle` object stores the resources for a locale. A `ResourceBundleFamily` object contains a collection of `ResourceBundles`, which groups the resources for an application. The application can switch languages, depending on the locale of the BlackBerry device user, without requiring new resource bundles.

Storing resources for a locale

You can use the BlackBerry® Integrated Development Environment to compile each resource bundle into a separately compiled .cod file. You can load the appropriate .cod files onto BlackBerry devices with the .cod files for the application. The BlackBerry IDE organizes resources in a hierarchy based on inheritance. If you do not define a string in a locale, a string from the next closest locale is used.

Files required for localization

File required for localization	Description	Example
Resource header	This file defines descriptive keys for each localized string. When the BlackBerry® Integrated Development Environment builds a project, it creates a resource interface with Resource appended to the .rrh file name. For example, if you create `AppName.rrh`, the interface name is `AppNameResource`.	AppName.rrh
Resource content (root locale)	This file maps resource keys to string values for the root (global) locale. It has the same name as the resource header file.	AppName.rrc
Resource content (specific locales)	This file maps resource keys to string values for specific locales (language and country). Resource files have the same name as the resource header file, but these are followed by an underscore (_) and the language code, and then, optionally, an underscore (_) and country code. Two-letter language and country codes are specified in ISO-639 and ISO-3166, respectively.	AppName_en.rrc AppName_en_GB.rrc AppName_fr.rrc
Initialization	This file initializes the resource bundle mechanism. This file is required only when you compile resources as a separate project.	init.java

Add localization support

Task	Steps
Add resource header files.	1. In the BlackBerry® Integrated Development Environment, on the **File** menu, click **New**. 2. In the **File name** field, type a file name. 3. Click **Browse**. 4. Select the folder that contains the file. 5. Click **OK**. 6. In the field, type the package name, for example, **com.rim.samples.docs.countryinfo**. 7. Click **OK**. 8. Click **Yes**. 9. Leave the file that appears in the text editor empty except for the package statement. 10. To add the file to your project, right-click the file in the right pane. 11. Click **Insert into project**.

Task	Steps
Add resource content files.	Create resource content files in the folder where the .java file exists. For example, in the folder that contains CountryInfo.java, create CountryInfo.rrc (root locale), CountryInfo_en.rrc (English), and CountryInfo_fr.rrc (French). 1. On the **File** menu, click **New**. 2. Type a file name and location. 3. Click **OK**. 4. Click **Yes**. 5. Leave the file empty. 6. To add the .rrc file to the application project, right-click the .rcc file in the right pane. 7. Click **Insert into project**.
Add resources.	1. In the BlackBerry IDE, double-click a resource header file. 2. Add resource keys: • To add one value per key: On the **Root** tab, type resource keys and values for each string or string array in your application. • To add multiple values per key: In the resource editor, right-click a resource and click **Convert to Multiple Values**. Add one or more values to the array. 3. To specify a different text string in other locales, select the tab for a locale, such as **fr** for the French language. 4. In the **Value** cell for the resource, type the text string for the locale. If you do not define a value for a resource in a particular locale, the value for the root locale is used. You can type unicode characters directly into the **Value** cell. Visit **http://www.unicode.org** for more information about unicode characters.
Display a localized application title on the Home screen.	If you do not provide a resource for the application title, the application uses the value you type in the **Title** field on the **Application** tab of the Project Properties window as the application title. 1. In the resource editor, add a resource for the application title, such as APPLICATION_TITLE. 2. Type a value for this resource in each locale that you support. 3. In the BlackBerry IDE, right-click the application project, and then click **Properties**. 4. On the **Resources** tab, select the **Title Resource Available** option. 5. From the **Resource Bundle** drop-down list, select the resource header file name to use for this application. 6. From the **Title Id** drop-down list, select the resource to use for the application title, such as APPLICATION_TITLE. 7. From the **Description Id** drop-down list, select a description ID.

Code sample: Storing text strings in separate resources for locales

Example: CountryInfo.java

```
/**
 * CountryInfo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
```

```
package com.rim.samples.docs.countryinfo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

/* This sample demonstrates how to store text strings in separate resource
   files for specific locales rather than providing text strings directly
   in the code. In your source code, you retrieve the string from the resource
   to display the appropriate text for the user locale. */
public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
    }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
    }
}

final class HelloWorldScreen extends MainScreen implements CountryInfoResource {
    private  InfoScreen  _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
            CountryInfoResource.BUNDLE_ID, CountryInfoResource.BUNDLE_NAME);

    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
                LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new ObjectChoiceField(
                _resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
    }

    public boolean onClose() {
        Dialog.alert(_resources.getString(CLOSE));
        System.exit(0);
        return true;
    }

    private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
        public void run() {
            select = choiceField.getSelectedIndex();
            _infoScreen = new InfoScreen();
            UiApplication.getUiApplication().pushScreen(_infoScreen);
        }
    };
```

```
    private MenuItem _closeItem = new MenuItem(_resources, MENUITEM_CLOSE,
            200000, 10) {
        public void run() {
            onClose();
        }
    };

    protected void makeMenu( Menu menu, int instance ) {
        menu.add(_viewItem);
        menu.add(_closeItem);
    }

    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
                    _resources.getString(FIELD_POP), null, 20, Field.READONLY);
            BasicEditField langField = new BasicEditField(
                    _resources.getString(FIELD_LANG), null, 20, Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
                    _resources.getString(FIELD_CITIES), null, 50, Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
                langField.setText(_resources.getString(FIELD_US_LANG));
                citiesField.setText(_resources.getString(FIELD_US_CITIES));
            } else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));
                langField.setText(_resources.getString(FIELD_CHINA_LANG));
                citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
            } else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));
                popField.setText(_resources.getString(FIELD_GERMANY_POP));
                langField.setText(_resources.getString(FIELD_GERMANY_LANG));
                citiesField.setText(
                    _resources.getString(FIELD_GERMANY_CITIES));
            }
        }
    }
}
```

Retrieve strings from a resource file

Task	Steps

Retrieve the resource bundle.	When the BlackBerry Integrated Development Environment builds your project, it creates an interface for each .rrh resource file.
	1. Import the interfaces that the BlackBerry IDE creates.
	`import com.rim.samples.docs.resource.CountryInfoResource;`
	2. Create a `ResourceBundle` object to contain the localized resources, such as strings, for an application.
	`private static ResourceBundle _resources = ResourceBundle.getBundle(CountryInfoResource.BUNDLE_ID, CountryInfoResource.BUNDLE_NAME);`
	3. To retrieve the appropriate bundle family, invoke `getBundle()`.

Create menu items using resources.	> Use the `MenuItem` constructor that accepts a resource bundle and a resource instead of a `String` for the name of the menu item.
	`private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {`
	`public void run() {`
	`select = choiceField.getSelectedIndex();`
	`_infoScreen = new InfoScreen();`
	`UiApplication.getUiApplication().pushScreen(_infoScreen);`
	`}`
	`}`

Replace text strings with the appropriate resources.	1. For each field that appears on the main screen, replace the text string with the appropriate resource.
	2. Invoke `getString()` or `getStringArray()` to retrieve the string for the appropriate language.
	`LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE), LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);`
	`add(new RichTextField(_resources.getString(FIELD_TITLE)));`
	`String choices[] = _resources.getStringArray(FIELD_COUNTRIES);`
	`choiceField = new ObjectChoiceField(_resources.getString(FIELD_CHOICE), choices);`

Code sample: Retrieving strings from a resource file

Example: CountryInfo.java (with localization support)

```
/**
 * CountryInfo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.countryinfo;

import net.rim.device.api.ui.*;
```

```
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

/* This sample demonstrates how to store text strings in separate resource
   files for specific locales rather than providing text strings directly
   in the code. In your source code, you retrieve the string from the resource
   to display the appropriate text for the user locale. */
public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
    }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
    }
}

final class HelloWorldScreen extends MainScreen implements CountryInfoResource {
    private  InfoScreen  _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
            CountryInfoResource.BUNDLE_ID, CountryInfoResource.BUNDLE_NAME);

    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
                LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new ObjectChoiceField(
                _resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
    }

    public boolean onClose() {
        Dialog.alert(_resources.getString(CLOSE));
        System.exit(0);
        return true;
    }

    private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
        public void run() {
            select = choiceField.getSelectedIndex();
            _infoScreen = new InfoScreen();
            UiApplication.getUiApplication().pushScreen(_infoScreen);
        }
    };

    private MenuItem _closeItem = new MenuItem(_resources, MENUITEM_CLOSE,
            200000, 10) {
        public void run() {
```

```
            onClose();
        }
    };

    protected void makeMenu( Menu menu, int instance ) {
        menu.add(_viewItem);
        menu.add(_closeItem);
    }

    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
                    _resources.getString(FIELD_POP), null, 20, Field.READONLY);
            BasicEditField langField = new BasicEditField(
                    _resources.getString(FIELD_LANG), null, 20, Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
                    _resources.getString(FIELD_CITIES), null, 50, Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
                langField.setText(_resources.getString(FIELD_US_LANG));
                citiesField.setText(_resources.getString(FIELD_US_CITIES));
            } else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));
                langField.setText(_resources.getString(FIELD_CHINA_LANG));
                citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
            } else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));
                popField.setText(_resources.getString(FIELD_GERMANY_POP));
                langField.setText(_resources.getString(FIELD_GERMANY_LANG));
                citiesField.setText(
                    _resources.getString(FIELD_GERMANY_CITIES));
            }
        }
    }
}
```

Manage resource files for application suites

When creating a suite of applications, organize the resources into separate projects for each locale.

Task	Steps
Create resource projects.	1. Open the BlackBerry® Integrated Development Environment. 2. Create a project for each resource bundle (locale), including the root locale. 3. Give the projects for each locale the same name as the project for the root locale, followed by a double underscore (__), the language code, and, optionally, an underscore (_) followed by the country code. For example, if the root locale project is named **com_company_app**, the projects for each locale would be named **com_company_app__en**, **com_company_app__en_GB**, and **com_company_app__fr**.
Specify output file names.	The output file names for all resource locale projects must be the same as for the root locale, followed by a double underscore and the appropriate language and country codes. 1. Right-click the project, and then click **Properties**. 2. On the **Build** tab, in the **Output file name** field, type a name for the compiled file, without a file name extension.

Create an initialization file.	> The BlackBerry IDE provides a built-in initialization mechanism, so that you only need to create an empty initialization class with an empty `main()`.
	`package com.rim.samples.device.resource;`
	`import net.rim.device.api.i18n.*;`
	`public class init {`
	`public static void main (String[] args) { }`
	`}`

Add files to appropriate resource projects.	If you support a large number of locales, create a single library project for all resource header (.rrh) files and set the project type to **Library**. For each resource locale in this project, define a dependency between the projects. 1. Create one .rrh file for each application. 2. Add the .rrh files to the project for each application. 3. Add the .rrh files to each resource project. 4. Create one resource content (.rrc) file for each application. 5. Create one .rrc file for each supported locale. 6. In each resource project, right-click each .rrh file, and then click **Properties**. 7. Select **Dependency only**. **Do not build**. 8. Add the .rrc files to the projects for the appropriate locales.

17

Testing applications

Testing applications using the BlackBerry IDE
Testing applications using BlackBerry devices
Debugging applications

Testing applications using the BlackBerry IDE

After you develop and compile your application, you should test it on the BlackBerry® device. The most common first step is to set up the BlackBerry Integrated Development Environment to use a BlackBerry device simulator for testing. The BlackBerry device simulators run the same Java™ code as the live BlackBerry devices, so they provide an environment for testing how applications will function on a live BlackBerry device. Each version of the BlackBerry Java Development Environment comes with the BlackBerry device simulators that are available when Research In Motion (RIM) released that version of the BlackBerry JDE. You can download additional device simulators as they are available on the BlackBerry Developer Zone at http://www.blackberry.com/developers/index.shtml.

Use the BlackBerry device simulator to test synchronizing data with the BlackBerry Desktop Software

The following instructions assume the BlackBerry® Desktop Software exists on your computer. See the *BlackBerry Integrated Development Environment Online Help* for information on starting the BlackBerry device simulator when you run an application in the BlackBerry IDE.

Task	Steps
Set the connection.	1. Connect a null modem cable between COM port 1 and COM port 2 on your computer. 2. In the BlackBerry IDE, on the **Edit** menu, click **Preferences**. 3. In the Preferences window, click the **Basic** tab. 4. Select the **Set serial port for device(s)** option. Type **1**. 5. Click **OK**.
Run the application.	> Build and run the application in the BlackBerry IDE.
Detect the simulator.	1. After the BlackBerry device simulator starts, start the BlackBerry Desktop Software. 2. In the BlackBerry Desktop Manager window, on the **Options** menu, click **Connection Settings**. 3. Click **Detect** to detect the BlackBerry device simulator.

Testing applications using BlackBerry devices

After testing your application on the BlackBerry® device simulator, load your application on a live BlackBerry device. When the application loads, you can open and test its functionality and performance. For debugging purposes, attach your BlackBerry device to the BlackBerry Integrated Development Environment debugger to step through your application code. This can be useful if you are trying to identify a network issue, a Bluetooth® wireless technology issue, or other items that are difficult to simulate accurately.

If your application uses signed APIs, you might require code signing keys. See the *BlackBerry Java Development Environment Fundamentals Guide* for more information about code signing keys.

Connect the BlackBerry IDE to a BlackBerry device

1. Connect a BlackBerry® device to the computer.
2. Open the BlackBerry Integrated Development Environment.
3. Connect the BlackBerry IDE to the BlackBerry device:
   - To connect a BlackBerry device that uses a USB port, click **Debug** > **Attach to** > **Handheld** > **USB<*PIN*>**, where *PIN* is the PIN of a BlackBerry device that is connected to the computer.
   - To connect a BlackBerry device that uses a serial port, click **Debug** > **Attach to** > **Handheld** > **COM <*n*>**, where *n* is the serial port that your BlackBerry device connects to. To make this connection, you must install the Java™ Communications API Version 2.0, which is available at http://java.sun.com/products/javacomm/. You do not need this API if you connect the BlackBerry device through a USB port.

Debugging applications

When you connect a BlackBerry® device to a computer to perform testing and optimization of code, run applications on the BlackBerry device and use the BlackBerry Integrated Development Environment debug tools.

Use breakpoints

In the breakpoints pane, perform any of the following tasks:

Task	Steps
Execute code and print values to the Output window when code execution reaches a Breakpoint.	> In the **Execute when hit** field, type an expression. For example, type **System.out.println(foo)**.
Open the source code at a set breakpoint.	> In the **Resume if true** field, type an expression. When application execution reaches the breakpoint, application execution resumes if the expression evaluates to true.
Stop the application after a specific number of iterations through a breakpoint.	> In the **Iteration** field, type a positive integer. When you start debugging, execution stops when the number of iterations through a breakpoint equals the number you typed.
Stop the application when a condition is true.	> In the **Condition** field, type a Boolean expression, such as x==100. The **Hits** field calculates the number of times the application stops at a breakpoint when the Condition is true or the **Condition** field is empty.

Task	Steps
Remove a breakpoint.	1. Open the source file. 2. In the Edit window, click the line of code that contains the breakpoint you want to remove. 3. On the **Debug** menu, click **Breakpoint** > **Delete Breakpoint at Cursor**.
Remove all breakpoints.	1. On the **View** menu, click **Breakpoints**. 2. In the breakpoints pane, click **Delete All**.

One debugging method is to start by setting only a few breakpoints at critical sections of your code, and then gradually set breakpoints at shorter intervals. To identify the problem, after the application pauses at a breakpoint, use debugging tools to view various application processes and statistics.

Debug an application in the BlackBerry IDE

1. Copy the application .cod, .csl, .cso, .debug, and .jar files into the BlackBerry® device simulator root directory:
   `C:\Program Files\Research In Motion\BlackBerry JDE 4.2.0\simulator`
2. Open the BlackBerry Integrated Development Environment.
3. Add and set up a workspace for your application.
4. Add your application to the workspace.
5. Start any simulators that your application requires.

   Without access to a BlackBerry Enterprise Server, you need the BlackBerry MDS™ Simulator to simulate browser traffic, HTTP/TCP connections to third-party applications, and push functionality. To debug applications that send and receive messages between a BlackBerry device simulator and a computer email application, use the BlackBerry email server simulator. See the *Simulator Guide* for more information.
6. On the **Debug** menu, click **Go**. The BlackBerry IDE builds all active projects in the workspace and loads the applications in the BlackBerry device simulator.
7. In the BlackBerry device simulator window, test the relevant sections of code.
8. On the **Debug** menu, click **Break Now**.
9. To resume debugging your application, on the **Debug** menu, click **Continue**.
10. To stop debugging your application, in the BlackBerry device simulator, on the **File** menu, click **Quit**.

Manage a debugging session

Task	Steps
Continue a debugging session.	> To resume running the application, on the **Debug** menu, click **Continue**.
End a debugging session in the BlackBerry® device simulator.	> In the BlackBerry device simulator, on the **File** menu, click **Quit**.
End a debugging session in the BlackBerry Integrated Development Environment.	1. In the main window, on the **Debug** menu, click **Stop Debugging**. 2. Read the warning message. 3. If you agree, click **Yes**.
Interrupt a debugging session without stopping an application.	> In the main window, on the **Debug** menu, click **Break Now**.

Locate an error in the source code

1. In the Output window, double-click the error message.
2. Perform one of the following tasks:

Task	Steps
View the next error.	> Press **F4**.
Return to the previous error.	> Press **SHIFT+F4**

Run an application to the insertion point

1. In the Edit window, click the line of code at which you want to stop the application.
2. In the **Debug** menu, click **Run to Cursor**.

Tip: To stop execution at a specific location, set a breakpoint on a line of code.

Debug an application on a BlackBerry device

To perform testing and optimization for applications on a BlackBerry® device that is in the cradle, use the BlackBerry Integrated Development Environment debugging tools.

System requirements

- To connect the BlackBerry® Integrated Development Environment to a BlackBerry device using a serial port, install the Java™ Communications API Version 2.0.
- To connect the BlackBerry IDE to a BlackBerry device using a USB port, install the BlackBerry Desktop Software Version 3.5.1 or later.

Install .debug files on your computer

To debug applications using a BlackBerry® device, the .debug files in the BlackBerry Integrated Development Environment must match the software version number of the BlackBerry device. BlackBerry device simulator packages contain .debug files for specific BlackBerry devices.

1. Download the BlackBerry device simulator package for your BlackBerry device software version number from the BlackBerry Developer Zone at:

   http://blackberry.com/developers/downloads/simulators
2. Connect a BlackBerry device to the computer. See "Connect the BlackBerry IDE to a BlackBerry device" on page 246 for more information about connecting a BlackBerry device to a computer.
3. In the BlackBerry IDE, on the **Edit** menu, click **Preferences**.
4. Click the **Debug** tab.
5. Click the **Other** tab.
6. In the **Handheld debug file location** field, type the location of the downloaded .debug files. The .debug files exist in the Debug directory of the BlackBerry device simulator package installation directory.

Load applications on a BlackBerry device

For development and testing purposes, use JavaLoader.exe to load applications onto the BlackBerry® device.

1. Exit the BlackBerry Desktop Software.
2. Connect the BlackBerry device to the computer.
3. At a command prompt, switch to the BlackBerry Java™ Development Environment bin folder.
4. Type the following command:

```
JavaLoader [-usb] [-pport] [-bbps] [-wpassword] load file
```

Parameter	Description
`port`	The serial port to which the BlackBerry device connects (default is COM1), or a BlackBerry device PIN if the BlackBerry device connects to a USB port. You must also specify the **-usb** option.
`bps`	The bit rate speed to the serial port (default is 115200).
`password`	If a password is set, the password for the BlackBerry device.
`file`	Specifies one or more .cod files to load onto the BlackBerry device.

Note: BlackBerry device users should use the BlackBerry Desktop Software to load applications onto their BlackBerry devices.

Connect the BlackBerry IDE to a BlackBerry device

Perform one of the following tasks:

Task	Steps
Connect to a BlackBerry® device that uses a serial port connection.	On the **Debug** menu, click **Attach to** > **Handheld** > **COM <n>**, where <n> is the serial port to which your BlackBerry device connects.
Connect to a BlackBerry device that uses a USB port connection.	On the **Debug** menu, click **Attach to** > **Handheld** > **USB (<PIN>)**, where <PIN> is the PIN of the BlackBerry device.

You can now run your applications on the BlackBerry device and use the BlackBerry Integrated Development Environment debugging tools to test and optimize your application.

Step through lines of code in an application

In the main window, on the **Debug** menu, perform any of the following tasks:

Task	Steps
Step over a method call.	The BlackBerry® Integrated Development Environment debugger moves to the next line of code. If the source line is a method call, the application runs the entire method without stepping through the individual method instructions.
Step through method instructions.	The BlackBerry IDE debugger moves to the next line of code. If the source line is a method call, the application stops just before running the first statement of the method.
Step out of method instructions.	The BlackBerry IDE debugger moves to the next line of code. If the source line is part of a method, the application runs the remaining lines of the method and returns control to the caller of the method.

For example, to step into function "f" in the following line of code f(g(x)), perform the following actions:

1. Click **Step Into** to run the application into "g."
2. Click **Step Out** to return the application to the line of code.
3. Click **Step Into** again to run the application into function "f."

View statistics to locate memory leaks

To locate memory leaks, use the Memory Statistics tool with the Objects tool. Begin by using the Memory Statistics tool to retrieve information on the memory usage of your application. The Memory Statistics tool identifies the number of objects in memory, while the Objects tool displays detailed information for each object.

The Memory Statistics tool displays the statistics on the number of objects and bytes in use for object handles, RAM, and flash memory.

Locate a memory leak

1. Set two or more breakpoints in your code.
2. Open the BlackBerry Integrated Development Environment.
3. On the **Debug** menu, click **Go**. The application runs to the first breakpoint.
4. In the main window, on the **View** menu, click **Memory Statistics**.
5. In the memory statistics pane, click **Refresh**.
6. Click **Snapshot**.
7. On the **Debug** menu, click **Continue**. The application runs to the second breakpoint.
8. In the memory statistics pane, click the **Refresh** tab.
9. Click **Compare to Snapshot**.

10. Repeat steps 1 through 8, setting breakpoints closer together until they converge on the memory leak.

Display objects in memory to locate object leaks

Object leaks can cause the JVM to run out of flash memory, which forces a BlackBerry® device to reset.

Display format

The Name column displays each process in the following format: `process_name(process_id): status`

where status is one of the following: Add, Delete, Referenced by code, Referenced by static, Grouped, Persistent, or RAM.

Status	Description
Add or Delete	This status appears when you perform a **Compare to Snapshot** to indicate new or removed objects since the last snapshot.
Referenced by code or Referenced by static	This status appears when a code (a local variable) or static data member references the variable.

Use the Objects tool to locate a memory leak

1. In the BlackBerry® Integrated Development Environment, on the **Debug** menu, click **Go**.
2. On the **Debug** menu, click **Break Now**.
3. On the **View** menu, click **Objects**.
4. In the objects pane, click **GC**.
5. In the objects pane, click **Snapshot**.
6. On the **Debug** menu, click **Continue**.
7. Perform operations in the application that do not increase the number of reachable objects. For example, create a new contact and then delete it.
8. On the **Debug** menu, click **Break Now**.
9. In the objects pane, click **GC**.
10. Click **Compare to Snapshot**.

    The objects pane displays the number of objects deleted and added since the previous snapshot. If the number of objects added is not the same as the number of objects deleted, you might have an object leak. To narrow new objects, use the **Type**, **Process**, and **Location** filters located at the top of the objects pane.
11. To save the contents of the objects pane to a comma separated values (.csv) file, click **Save**.

Show references to or from an object

> In the objects pane, right-click an object, and then click **Show References to** or **Show References From**.

The object view narrows to show only the objects that have references to or from this object. Use the **Forward** and **Back** buttons to move back and forth through the reference chain.

Right-click an object, and then click **Show Recursive References To @nnnnnnnn** to display all objects that reference the selected object. An object can indirectly display another object through it.

Note: This operation might take a long time to complete.

Show the source code or static

Double-click **Code referencing @nnnnnnnn** or **Static referencing @nnnnnnnn** line to display that source code or static. Click **Forward** and **Back** to move back and forth through the reference chain.

View local variables

1. On the **View** menu, click **Locals**.
2. Perform one of the following tasks:

Task	Steps
View local variables and their current values in the context of the current thread.	> Click the **Locals** tab.
View local variable names and expressions at and around the executing line.	> Click the **Auto** tab.
View an expanded view of the current object (this).	> Click the **This** tab.
Evaluate expressions.	> Click the **Watch** tab.

View variable or expression information

Task	Steps
View a value for a variable.	1. Point the insertion point at a variable. 2. Press the **CTRL** key and click a variable.
View a value for an expression.	1. Press the **CTRL** key and click an expression.

View static data

The static data pane displays the static data members of the current class.

The following options are available when you right-click the static data pane:

- Change the display of the Value field.
- Set a watch on a variable.
- View the source code of a variable's defining class.
- When the application modifies an item, stop the application.

Evaluate (watch) Java expressions

The Watch pane lets you specify variables and expressions to watch continuously while debugging your application.

1. Right-click the **Watch** pane.
2. Perform one of the following tasks:

Task	Steps
Set a new watch.	> Click **Add Watch**.
Remove a watch.	> Click **Delete**.
Remove all watches.	> Click **Delete All**.

See "View threads" on page 253 for more information about viewing the format of threads appearing in the Watch pane.

See "View the data members of a process" on page 254 for more information about viewing the format of the processes that appear in the Watch pane.

View threads

The threads pane displays all threads running on the BlackBerry® device. The most recent thread appears yellow.

To view the source code in the text edit pane, double-click a thread. The BlackBerry Integrated Development Environment marks the line in the source code that starts the thread with an arrow.

Thread format

The Thread column displays each thread in the following format:

```
name(pid): status
```

where:

- name is the name of the process that starts the thread
- pid is the ID of the process that starts the thread
- status is one of the following:

Status	Description
running	thread is running
sleeping	thread is calling `Thread.sleep()`
waiting for notify	thread is calling `Object.wait()`
acquiring lock	thread is executing a "synchronized" statement and is being forced to wait

Make a thread current

When you make a thread current, the Call Stack changes to display the calls for the thread. Other windows might display current information relating to the new current thread.

1. From a variable window, right-click a thread.
2. Select **Make thread current**.

Expand objects

In the threads pane, the following fields indicate the status of an object:

- thread that currently owns the object (Thread owning lock: @nnnnnnnn)
- thread that calls `Object.wait()` (Thread waiting for notify: @nnnnnnnn)
- thread that attempts to enter a synchronized block for the object (Thread acquiring lock: @nnnnnnnn)

Note: Threads, and objects whose threads are deadlocked, also display in the following panes: objects, local variables, watch, static data, processes, and locks. To update the context of the selected thread in all the debugging panes, right-click in the threads pane, and then click **Make current**.

View the data members of a process

The processes pane lists all the processes that are currently running in the BlackBerry® Integrated Development Environment. You can expand each process to view its data members.

The Process column displays each process in the format `process_name(process_id)`.

> To view data members, in the **Process** column, expand a process.

View the call stack

The call stack pane displays the calling methods at the current point of execution.

View the source code of a calling method

1. Right-click a method.
2. Click **Show Definition**.

The source file appears in the Edit window at the line of code that implements the class of the selected item. All BlackBerry® Integrated Development Environment panes update to reflect the new context.

Note: The first calling method that appears in the call stack pane is located at the bottom of the call stack.

View event logs

The event log pane displays all exception messages that the BlackBerry® Integrated Development Environment produces when you run an application in the BlackBerry device simulator or on a BlackBerry device. To identify an error that has occurred, use the event log pane to view the source code that caused the error message.

View the source of a logging message

> In the event log pane, on the **Build** tab, double-click the error message.

View classes

Select a subset of classes

Type the Class Name Prefix and press **ENTER**. For example, type `java.lang`. In the classes pane, classes that start with the string typed in the Class Name Prefix field appear.

1. In the classes pane, right-click a class.
2. Perform one of the following tasks:

Task	Step
> Display the source code that implements the selected class.	Click **Source code**.
> Set the BlackBerry® Integrated Development Environment to trigger a breakpoint when the code throws an object of the selected class.	Click **Break when exception thrown**.
> Set the BlackBerry IDE to trigger a breakpoint when an object of the selected class is instantiated.	Click **Break on new object**.

View the methods in a class

The methods pane displays all methods in a class. In the classes pane, double-click a class. The methods pane updates to display all methods in the selected class.

Optimize source code using the BlackBerry IDE profiler tool

To optimize your code, use the profiler tool of the BlackBerry® Integrated Development Environment. The profiler tool displays the percentage of time spent in each code area to the current point of execution.

Note: To improve the quality of results when you run the profiler tool, exit other Microsoft® Windows® applications.

Set profile options

1. In the profile pane, click **Options**.
2. On the **General** tab, set the following options:

Drop-down list	Option	Description
Method attribution	Cumulative	The profiler tool calculates the time spent executing bytecode in a method and all methods that the method invokes.
	In method only	The profiler tool calculates the time spent executing bytecode in that method only. The timer stops when a call is made to another method.

Drop-down list	Option	Description
Sort method by	Count	The profiler tool sorts methods in the profile pane by the number of times the application executed the item.
	Profiled data (select in "What to profile")	The profiler tool sorts methods in the profile pane by the data you choose to profile.
What to profile	Time (clock ticks)	The profiler tool considers execution time (measured in clock ticks).
	Number of objects created	The profiler tool considers the number of objects that the application created.
	Size of objects created	The profiler tool considers the size of the objects that the application created.
	Number of objects committed	The profiler tool considers the number of objects that the application committed.
	Size of objects committed	The profiler tool considers the size of the objects that the application committed.
	Number of objects moved to RAM	The profiler tool considers the number of objects that the application moved into memory.
	Size of objects moved to RAM	The profiler tool considers the size of the objects that the application moved into memory.
	User Counting	The profiler tool considers user counting.

3. Click the **Colors** tab to change the colors of the source code highlighting.

Generate profile data

1. Set a breakpoint at the start of the section of code that you want to profile.
2. Set a breakpoint at the end of the section of code that you want to profile.
3. On the **Debug** menu, click **Go**.
4. In the BlackBerry® device simulator, run the application. The debugger pauses the application when it reaches the first breakpoint.
5. On the **View** menu, click **Profile**.
6. In the profile pane, click **Options**.
7. Select the type of method attribution, a sorting method, and the type of information that you want to profile.
8. Click **OK**.
9. In the profile pane, click **Clear**. This action removes the profiler data and sets the running time of Java™ code to 0.
10. On the **Debug** menu, click **Go**.
11. In the BlackBerry device simulator, run the application. The debugger pauses the application when it reaches the second breakpoint.
12. If the profile pane is not visible, on the **View** menu, click **Profile**.

13. In the profile pane, click **Refresh**. This action retrieves all accumulated profile data from the JVM. This action does not clear Profiler data, so running an application again adds to the data.
    - Use profile views to view information about the section of code that you just ran.
14. Click **Save** to save the contents of the profile pane to a comma separated values (.csv) file.

View profile data

The profile pane has three views. Each view contains details about an item of execution (such as a method), the percentage of time that the application ran the item, and the number of times that the appplication ran the item.

Note: To view all accumulated data, click **Refresh**.

1. On the **View** menu, click **Profile**.
2. Click one of the following tabs:

View	Description
Summary	The Summary view displays general statistics about the system and the garbage collection process. It displays the percentage of time that the JVM spends idle, executing code, and performing quick and full garbage collection processes. The Percent column displays the percent of total JVM running time, including idle and collection times.
Methods	The Methods view displays a list of modules, sorted either by the information that you are profiling or by the number of times that the application executed each item. • Expand the **All** item to see a list of all methods. • Expand a specific module to see only its methods. • Right-click a method, and then click **Profile Source** to view source lines in the Source view. • Right-click a method, and then click **Show Source** to view source code in the Edit window. In this view, the **Percent** column displays the percentage of JVM execution time only, not including idle and garbage collection times.
Source	The Source view displays the source lines of a single method. You can navigate through the methods that call, and are called by, that method. The Source view displays the following items: • a list of callers to the method, including the number of times that they make the call and the total time spent on these calls • a list of source lines for the method and the total time spent on these lines You can expand a source line to show individual bytecode. You can further expand any bytecode that corresponds to a method invocation to show the target(s) of the method invasion. Right-click a line and select **Show Source** to view the source code in the Edit window. Click **Back** and **Forward** to follow the history of methods that you have visited in the Source view.

Analyze code coverage

The Coverage tools display a summary of code that has run.

Run the Coverage tool

1. Set two or more breakpoints in your code.
2. Run the application to the first breakpoint.
3. On the **View** menu, click **Coverage**.
4. To reset the information to 0, in the coverage pane, click **Clear**.
5. Run the application to the next breakpoint.
6. To display the percentage of code that you ran since you clicked **Clear**, in the coverage pane, click **Refresh**.

The Coverage pane displays the percentage of code that you ran. It displays a nested view of packages, classes, and methods, with the percentage of code executed in each.

View source code

> In the coverage pane, double-click a method.

Green bars in the source code indicate that the source code ran, and red bars in the source code indicate that the source code did not execute.

Note: When you use the ternary if-else operator, the coverage tool displays accurate but misleading results. For example, your code might include the following statement:

```
a ? b : c;
```

if "a" is always true, then "c" will never execute; however, the coverage tool displays the statement as covered.

You can work around this by rewriting the code to avoid the ternary operator, as shown in the following code:

```
if( a ) {
   b;
} else {
    c;
 }
```

The short-circuit logical operators && and || exhibit the same behavior.

Approve HTTP connections

The BlackBerry® device includes built-in security features to prevent third-party applications from sending or receiving data without the knowledge of the BlackBerry device user. When a third-party application attempts to open a connection, a dialog box prompts the BlackBerry device user to turn the connection on or off. To test this functionality on the BlackBerry device simulator, turn on the security feature of the BlackBerry device simulator.

This security feature causes network applications to hang if an application makes an HTTP connection from the main thread.

When the main thread manages connection requests, the UI cannot initiate a dialog box to prompt the BlackBerry device user to approve the connection attempt. The application locks because the connection request cannot complete until the BlackBerry device user approves it.

To resolve this issue, you must put the network connection request on a separate thread from the main thread so that it does not interfere with the process of the main thread.

Start the BlackBerry email simulator

The BlackBerry® email simulator lets you send and receive email messages between the BlackBerry device simulator and an actual email account, without a BlackBerry Enterprise Server. To retrieve the BlackBerry email simulator, download the BlackBerry Email and MDS Services Simulator Package from the BlackBerry Developer Zone web site:

www.blackberry.com/developers

1. On the **Start** menu, click **Programs** > **Research In Motion** > **BlackBerry Email and MDS Services Simulators 4.1.2** > **ESS**.
2. Select one of the following modes:

Mode	Description
Standalone mode	ESS stores messages on the local file system and communicates directly with the email client. You do not require a POP3 or SMTP server. ESS can communicate with any email client that supports POP3 and SMTP communication. The email client account must have the POP3 server set to localhost on port 110 and the SMTP server set to localhost on port 25.
Connected mode	ESS polls the specified POP3 email server for incoming messages and uses the specified SMTP server to send messages. The Connected mode requires a Valid POP3 and SMTP server.

3. If you select Standalone mode, click **Clean FS** to erase ESS messages that exist on the local file system.
4. If you select Connected mode, type information in the following fields:

Field	Description
Outgoing	host name of the SMTP server that your email account uses
Incoming	host name of the POP3 server that your email account uses
User name	user name with which to connect to your email account
Password	password with which to connect to your email account
Poll inbox	frequency with which the BlackBerry device simulator checks your email inbox folder for new messages

5. Type information in the following fields:

Field	Description
Name	name to display in outgoing messages from the BlackBerry device simulator
Email	email address to display in outgoing messages from the BlackBerry device simulator
PIN	PIN that the BlackBerry device simulator uses (default is 21000000)

6. Click **Launch**.
7. Click **Load Test** to select a message inside the associated **Inbox** and send it a number of times to a BlackBerry device.

   One or more email messages must exist inside the inbox of the email account associated with the BlackBerry email simulator for the load test functionality to work.

8. Check the command prompt window for detailed information about ESS startup, including any login errors.

Working with compiled applications

When you build a project using the BlackBerry® Integrated Development Environment, the BlackBerry IDE compiles your source files into Java™ bytecode, performs preverification, and creates a single .cod file and .jad file for an application.

If an application contains more than 64 KB of bytecode or resource data, the BlackBerry IDE creates a .cod file that contains sibling .cod files. To determine if a .cod file contains sibling .cod files, extract the contents of the .cod file. Any .cod files within the original .cod file are the sibling files.

To identify modules that an application requires but are not provided with it, examine the application descriptor (.jad) file RIM-COD-Module-Dependencies attribute. See "Appendix: BlackBerry application .jad files" on page 281 for more information about BlackBerry application .jad file properties.

Load and remove applications

To load, remove, or save .cod files when testing applications, use the JavaLoader.exe tool. For production applications, use the BlackBerry® Desktop Software.

Note: You must load applications with dependencies in the correct order. If project A is dependent on project B, load the project B .cod file before loading the project A .cod file.

1. Connect the BlackBerry device to the computer.
2. Open a command prompt, and navigate to the location of the JavaLoader.exe file.
3. Perform one of the following tasks:

Task	Steps
Load an application .cod file on the BlackBerry device.	> Issue a command using the following format: `javaloader [-u] load <.cod file>` For example: `javaloader.exe -u load MyApplication.cod` JavaLoader loads the .cod files listed in the .jad file on the BlackBerry device and stores the .cod files in a CodeModuleGroup.
Load application .cod files listed in the same .jad file on the BlackBerry device.	> Issue a command using the following format: `javaloader [-u] load <.jad file>` For example: `javaloader.exe -u load MyApplication.jad` Javaloader loads the .cod files listed in the .jad file onto the BlackBerry device.
Remove an application .cod file from the BlackBerry device.	> Issue a command using the following format: `javaloader [-u] erase [-f] <module>` For example: `javaloader.exe -u erase MyApplication`
Remove application .cod files listed in the same .jad file from the BlackBerry device.	> Issue a command using the following format: `javaloader delete <.jad file>` For example: `javaloader.exe delete MyApplication.jad`
Remove application .cod files stored in the same CodeModuleGroup from the BlackBerry device.	> Issue a command using the following format: `javaloader delete [-g] <module>` For example: `javaloader.exe delete -g MyApplication`

Task	Steps
Save an application .cod file from the BlackBerry device to your computer.	> Issue a command using the following format: `javaloader save <.cod file>` For example: `javaloader.exe save MyApplication.cod`
Save application .cod files listed in the same .jad file from the BlackBerry device to your computer.	> Issue a command using the following format: `javaloader save <.jad file>` For example: `javaloader.exe save MyApplication.jad`
Save application .cod files stored in the same CodeModuleGroup from the BlackBerry device to your computer.	> Issue a command using the following format: `javaloader save [-g] <module>` For example: `javaloader.exe save -g MyApplication`

View application information

1. Connect the BlackBerry® device to the computer.
2. Open a command prompt and navigate to the location of **JavaLoader.exe**.
3. Issue one of the following tasks:

Task	Steps
Retrieve Name, Version, Size, and Date created information for a .cod file.	> Issue a command using the following format: `javaloader info <.cod file>` For example: `javaloader.exe info MyApplication.cod`
Retrieve a list of .cod files that a .cod file requires to run.	> Issue a command using the following format: `javaloader info [-d] <.cod file>` For example: `javaloader.exe info -d MyApplication.cod`

18

Packaging and distributing applications

Preverify applications
Determine if your code requires signatures
Register to use RIM controlled APIs
Request code signatures
Distributing applications over the wireless network
Distributing applications with the BlackBerry Desktop Software

Preverify applications

To reduce the amount of processing the BlackBerry® device performs when you load your application, partially verify your classes.

> Issue a command from the command line in the following format:

```
preverify.exe [-d] output -classpath <directory> input; <directory>
```

To preverify .cod files, you may also use the BlackBerry device simulator. See the *Simulator Guide* for more information about the BlackBerry device simulator.

Determine if your code requires signatures

Research In Motion (RIM) tracks the use of sensitive APIs in the BlackBerry® Java™ Development Environment for security and export control reasons.

> Locate the item in the API Reference. If the item has a lock icon or is noted as "signed", your application requires a signed key or signature, which RIM provides, before you can load the application .cod files onto a BlackBerry device.

Controlled APIs

Three categories of Research In Motion (RIM) Controlled APIs exist: Runtime APIs, BlackBerry® Application APIs, and BlackBerry Cryptography APIs. See the *API Reference* for more information about all RIM controlled APIs.

You can run applications that use controlled APIs in the BlackBerry device simulator without code signatures; however, you must request code signatures from RIM before you can load these applications onto BlackBerry devices.

Note: To test and debug your code before receiving code signatures, use the BlackBerry device simulator. Code must have code signatures for loading onto BlackBerry devices.

If you use any of the following BlackBerry API packages, your application requires code signatures before you can load it on a BlackBerry device:

- net.rim.blackberry.api.browser
- net.rim.blackberry.api.invoke
- net.rim.blackberry.api.mail
- net.rim.blackberry.api.mail.event
- net.rim.blackberry.api.menuitem
- net.rim.blackberry.api.options
- net.rim.blackberry.api.pdap
- net.rim.blackberry.api.phone
- net.rim.blackberry.api.phone.phonelogs
- net.rim.device.api.browser.field
- net.rim.device.api.browser.plugin
- net.rim.device.api.crypto.*
- net.rim.device.api.io.http
- net.rim.device.api.notification
- net.rim.device.api.servicebook
- net.rim.device.api.synchronization
- net.rim.device.api.system

Register to use RIM controlled APIs

1. Complete the registration form on the BlackBerry® Developer Zone at https://www.blackberry.com/JDEKeys/.
2. Save the .csi file that Research In Motion (RIM) sends to you in an email message. The .csi file contains a list of signatures and your registration information.

   If the BlackBerry Signing Authority Tool administrator does not provide you with the .csi file or the Client PIN and you are an ISV partner, contact your ISV Technical Partnership Manager. If you are not an ISV partner, send an email message to jde@rim.com.
3. Double-click the .csi file.
4. If a dialog box appears that states that a private key cannot be found, follow the instructions to create a new key pair file.
5. In the **Registration PIN** field, type the **PIN** that RIM provided.

6. In the **Private Key Password** field, type a password of at least eight characters. The private key password protects your private key. If you lose this password, you must register again with RIM. If this password is stolen, contact RIM immediately.
7. Click **Register**.
8. Click **Exit**.

Restricted access to code signatures

The BlackBerry® Signing Authority Tool administrator might place restrictions on your .csi file to limit your access to code signatures. To request changes to these restrictions, contact your system administrator.

.csi file restriction	Description
# of Requests	Sets the maximum number of requests you can make using a particular .csi file. When you reach the maximum number of requests, the .csi file becomes invalid. To make new code signature requests, you must apply for a new .csi file. Although an administrator can permit an infinite number of requests, the number of requests is often set to a finite number for security reasons.
Expiry Date	Sets the expiry date for your .csi file. After the expiry date, you can no longer apply for code signatures with this .csi file. To make new signature requests, you must apply for a new .csi file.

Request code signatures

To perform this task, you must obtain a .csi file from Research In Motion (RIM). See "Register to use RIM controlled APIs" on page 264 for more information about obtaining a .csi file from RIM.

1. In Microsoft® Windows® Explorer, locate the .cod file for the application for which you are requesting a signature.
2. Make sure that a .csl file with the same name as the .cod file exists in the same folder as the .cod file. The BlackBerry® Integrated Development Environment compiler automatically generates the .csl file.
3. Double-click the .cod file to add it to the signature list. The signature list contains information on the .cod files that you want permission to access and are requesting signatures for.
4. Repeat steps 1 through 3 for each .cod file that you want to add to the signature list.
5. On the BlackBerry Signature Tool menu, click **Request**.

   The BlackBerry Signature Tool is part of the BlackBerry Java™ Development Environment installation. The BlackBerry JDE is available for download from the BlackBerry Developer Zone:

   http://www.blackberry.com/developers/
6. In the dialog box, type your private key password.
7. Click **OK**. The BlackBerry Signature Tool uses the private key password to append the signature to the request, and it sends the signature list of .cod files to the Web Signer application for verification. The Web Signer application installs when you install the BlackBerry Signing Authority Tool. See the *BlackBerry Signing Authority Tool - Password Based Administrator Guide* for more information about the Web Signer application.

Request code signatures using a proxy server

Task	Steps

Register signature keys using a proxy server.	You can register each .csi file only once.
	1. At the command prompt, browse to the BlackBerry® Signature Tool bin directory. For example:
	`C:\Program Files\Research In Motion\BlackBerry JDE 4.2.0\bin`
	2. Type the following command:
	`Java -jar -Dhttp.proxyHost=<myproxy.com> -Dhttp.proxyPort=80 SignatureTool.jar <SigKey>.csi`
	• `SigKey`: The name of each signature key (.csi) file. Use the following naming conventions for the keys: `client-RRT-*.csi, client-RBB-*.csi, client-RCR-*.csi.`
	• `Dhttp.proxyHost`: The name or IP address of the proxy server.
	• `Dhttp.proxyPort`: The proxy server port number if you do not specify 80 as the default port number.
	3. Repeat step 2 for each .csi file that you want to register.

Sign an application using a proxy server.	1. At the command prompt, browse to the BlackBerry Signature Tool bin directory. For example:
	`C:\Program Files\Research In Motion\BlackBerry JDE 4.2.0\bin`
	2. Type the following command:
	`Java -jar -Dhttp.proxyHost=<myproxy>.com -Dhttp.proxyPort=80 SignatureTool.jar`
	3. In the File Selection window, select the .cod file(s) to sign.
	4. Click **Open**.

Request a replacement registration key

Your registration key and .csk file are stored together. If you lose the registration key or the .csk file, you cannot request code signatures.

> If you are an ISV partner and lose the .csk file, contact your ISV Technical Partnership Manager.

> If you are not an ISV partner, send an email message to jde@rim.com.

View signature status

1. Start the BlackBerry® Signature Tool.
2. Select a .cod file.
3. View the Status column:
   - For files the Web Signer has signed, the **Status** column contains **Signed**.
   - For files the Web Signer did not sign, the **Status** column contains **Failed**. The Web Signer might have rejected the .cod file because the private key password was typed incorrectly.

Distributing applications over the wireless network

Method	Description
User-initiated wireless pull	Developers can post their compiled applications to a public or private web site, and BlackBerry® device users can download the application over the wireless network by pointing the web browser on their BlackBerry devices to this URL. When a BlackBerry device user visits the URL, the web browser prompts the BlackBerry device user to install the application. If the BlackBerry device user accepts, the application downloads over the wireless connection and installs immediately.
Server-initiated wireless push	In an enterprise environment, the BlackBerry Enterprise Server administrator can push applications out to BlackBerry device users over the wireless network and enforce that the application installs. The administrator simply creates a new policy and indicates that the application is required. Once the policy is set on the BlackBerry Enterprise Server, the application is sent to the BlackBerry device users without the need for any actions on the part of the BlackBerry device users.

Distribute applications

Task	Steps
Deploy .jar files.	This task requires that a BlackBerry® device user download a .jar file to a BlackBerry device that connects to a BlackBerry Enterprise Server. > Make .jar files available for download. When BlackBerry device users use the BlackBerry Browser to download a .jar file, the BlackBerry MDS™ Data Optimization Service feature of the BlackBerry Enterprise Server converts the .jar file to a .cod file.
Deploy .cod files.	1. Set the required MIME type on the web server. • For .cod files, set the MIME type to **application/vnd.rim.cod**. • For .jad files, set the MIME type to **text/vnd.sun.j2me.app-descriptor**. • For .jar files, set the MIME type to **application/java-archive**. 2. Place the .cod and .jad files on the web server for the BlackBerry device users to download. By making .cod files available, you can make available applications to BlackBerry device users who do not access the network using a BlackBerry Enterprise Server.
Set .cod file dependencies.	If any of the required modules are not present, the BlackBerry Browser prevents the wireless installation of the application and lists the missing modules for the BlackBerry device user. > In the application descriptor (.jad) file, use the **RIM-COD-Module-Dependencies** attribute to specify modules that the application requires but that the application does not provide. For example, an application that requires the RIM XML library might use the following in the application descriptor: **RIM-COD-Module-Dependencies: net_rim_cldc, net_rim_xml**
Determine if a .cod file contains sibling .cod files.	The following information is required only if BlackBerry device users access applications using the BlackBerry Internet Service or a WAP gateway. > Extract the contents of the .cod file. Any .cod files within the original .cod file are the sibling files.
Deploy a .cod file with sibling .cod files to a BlackBerry device that is not connected to a BlackBerry Enterprise Server.	> Modify the application .jad file so that the file lists each sibling file individually.

Task	Steps

Extract sibling .cod files.	To ensure a BlackBerry device user does not override the original .cod file, on the web server, extract the sibling .cod files into a different directory than the directory where the original file exists.
	1. Unzip the original .cod file and extract the sibling .cod files.
	2. Place each sibling .cod file on a web server.
	3. In the .jad file, list the sibling .cod files separately. For each sibling file, create `RIM-COD-URL-<#>` and `RIM-COD-Size-<#>` parameters. Use the following naming convention for sibling .cod files: <*name of original .cod file*>-<*sequential number*>.
	• `RIM-COD-URL-<#>`: Create a `RIM-COD-URL-<#>` parameter for each sibling .cod file, and place the name of the sibling file to the right of this parameter. **#** is a number that starts at 1 and increases by 1 for each sibling file. Give each sibling .cod file the same name as the original .cod file, followed by **-<#>**.
	• `RIM-COD-Size-<#>`: Create a `RIM-COD-Size-<#>` parameter for each sibling .cod file, and place the size of the sibling file to the right of this parameter. **#** is the same number that is appended to the name of the sibling file. Place the `RIM-COD-Size-<#>` parameter immediately below the `RIM-COD=URL-<#>` parameter.
	The following example shows two sibling files. The developer names the sibling files **myApp-1.cod** and **myApp-2.cod**, after the original .cod file myAPP. The developer appends the .cod file extension to each sibling file name. The developer creates a `RIM-COD-Size-<#>` parameter for each sibling file.
	`Manifest-Version: 1.0`
	`MIDlet-Version: 1.0.0`
	`MIDlet-1: ,,`
	`RIM-COD-Module-Dependencies: net_rim_cldc`
	`MicroEdition-Configuration: CLDC-1.0`
	`RIM-COD-Module-Name: MyApp`
	`MIDlet-Name: My Application`
	`RIM-COD-URL: myApp.cod`
	`RIM-COD-Size: 55000`
	`RIM-COD-URL-1: myApp-1.cod`
	`RIM-COD-Size-1: 50000`
	`RIM-COD-URL-2: myApp-2.cod`
	`RIM-COD-Size-2: 25000`
	`MicroEdition-Profile: MIDP-1.0`

Distribute individual sibling .cod files.	Place each sibling .cod file onto a web server.

Distributing applications with the BlackBerry Desktop Software

Deployment method	Description
Application Loader tool of the BlackBerry® Desktop Manager	The Application Loader tool of the BlackBerry Desktop Manager lets you install third-party applications as well as updated system software for the BlackBerry device. It lets BlackBerry device users download applications on their computers and install them on their BlackBerry devices.
BlackBerry Application Web Loader	With the BlackBerry Application Web Loader, you can post your compiled application on a central web site and BlackBerry device users can install the application by using Microsoft® Internet Explorer® on their computers to visit the URL. When BlackBerry device users visit the web page, they are asked to connect their devices to their USB ports. The application is then installed using an ActiveX® control. The BlackBerry Application Web Loader provides a simple approach for installing applications from your desktop and does not require the BlackBerry device user to run the BlackBerry Desktop Manager.
Javaloader Command Line Tool	The BlackBerry Java™ Development Environment includes a command line tool called Javaloader.exe. The executable file exists in the BIN directory under the JDE installation. Javaloader can be used to quickly install and remove compiled application files on the BlackBerry device directly over the USB port and does not require any descriptor files or web pages. Javaloader can be useful when you are installing and removing your application frequently during testing and development;. however, Javaloader is not a tool that BlackBerry device users would use.

Create an application loader file

1. Create an .alx file for each application, and then distribute the .alx file with the .cod files to BlackBerry® device users. See the *Application Loader Online Help* for more information about .alx files.
2. In the BlackBerry Integrated Development Environment, select a project.
3. On the **Project** menu, click **Generate .alx file**.

Load an application on a specific BlackBerry device

1. Open a text editor.
2. Locate the .alx file for the application.
3. In the .alx file, make sure the series attribute in the **fileset** opening tag refers to the BlackBerry® device you want the application to load on.

   ```
   <fileset series="8700" Java="1.0">
   ```

   For more information about the series attribute, see **Platform.alx** located in the simulator directory of your BlackBerry Java™ Development Environment installation directory:

   Program Files\Research In Motion\BlackBerry JDE 4.2.0\simulator.
4. Make sure the **files** tag contains a reference to the .cod file for your application.

   ```
   <files>
   ```

```
        My_application.cod
    </files>
```

5. Update the **application**, **description**, and other tags to reflect the purpose of the .alx file.

```
<application id="Push only to 8700">
...
<description>This will push the COD only to 8700s</description>
```

Example: Load an application on a specific BlackBerry device

```
<loader version="1.0">
    <application id="Push only to 8700">
    <name>Alien</name>
    <description>This will push the COD only to 8700s</description>
        <version>2006.02.14.1838</version>
        <vendor>RIM</vendor>
        <copyright>Copyright (c) 2001-2005</copyright>
        <fileset series="8700" Java="1.0">
            <files>
                My_application.cod
            </files>
        </fileset>
    </application>
</loader>
```

Specify optional components

In most cases, you do not need to change the .alx files that the BlackBerry® Integrated Development Environment generates.

1. Open a text editor.
2. In the text editor, open the .alx file that you want to change.
3. To provide optional components for an application, in the .alx file, create a nested structure.

   The .alx file uses an XML format:

```
Sample .alx file
<loader version="1.0">
    <application id="com.rim.samples.device.httpdemo">
    <name>Sample Network Application</name>
        <description>Retrieves a sample page over HTTP connection.
        </description>
    <version>1.0</version>
    <vendor>Research In Motion</vendor>
    <copyright>Copyright 1998-2003 Research In Motion</copyright>
    <language langid="0x000c">
        <name>Application D'Échantillon</name>
        <description>Obtenir une page du réseau
    </description>
    </language>
    <fileset Java="1.0">
        <directory>samples/httpdemo</directory>
        <files>
            net_rim_httpdemo.cod
```

```
            net_rim_resource.cod
            net_rim_resource__en.cod
            net_rim_resource__fr.cod
         </files>
      </fileset>
      </application>
   </loader>
```

4. To define an explicit dependency on another application or library, use the `<requires>` tag.

Example: Sample .alx file for an application with a nested module

```
<loader version="1.0">
   <application id="net.rim.sample.contacts">
      <name>Sample Contacts Application</name>
   <description>Provides the ability to store a list of contacts.
      </description>
      <version>1.0</version>
      <vendor>Research In Motion</vendor>
      <copyright>Copyright 1998-2001 Research In Motion</copyright>
      <fileset Java="1.0">
         <directory>samples/contacts</directory>
         <files>
            net_rim_contacts.cod
            net_rim_resource.cod
            net_rim_resource__en.cod
            net_rim_resource__fr.cod
         </files>
      </fileset>
      <application id="net.rim.sample.contacts.mail">
         <name>Sample Module for Contacts E-Mail Integration</name>
         <description>Provides the ability to access the messaging
            application</description>
         <version>1.0</version>
         <vendor>Research In Motion</vendor>
         <copyright>Copyright 1998-2001 Research In Motion</copyright>
         <fileset Java="1.0">
            <directory>samples/contacts</directory>
            <files>
               net_rim_contacts_mail.cod
            </files>
         </fileset>
      </application>
   </application>
</loader>
```

Specify supported BlackBerry Device Software

Applications that use APIs available only on particular versions of the BlackBerry® Device Software should specify supported BlackBerry device versions using the `_blackberryVersion` attribute.

> Specify a range using the following rules:

- Square brackets [] indicate inclusive (closed) range matching.

- Round brackets () indicate exclusive (open) range matching.
- Missing lower ranges imply 0.
- Missing upper ranges imply infinity.

For example, [4.0,) indicates any version between 4.0 and infinity.

The following example prevents modules from loading on versions of the BlackBerry Device Software earlier than version 4.0.

```
<application id="<application_id>" _blackberryVersion="[4.0,)">
...
</application>
```

The following example provides alternative modules for different versions of the BlackBerry Device Software.

```
<application id="<application_id>">
   ...
   <fileset _blackBerryVersion="(,4.0)">
   ... modules for BlackBerry device software versions earlier than 4.0
   </fileset>
   <fileset _blackBerryVersion="[4.0,)">
   ... modules for BlackBerry device software versions 4.0 and later
   </fileset>
</application>
```

See "Appendix: .alx files" on page 277 for more information about .alx file elements.

A

Appendix: The command line compiler

Using the command line compiler

Using the command line compiler

The BlackBerry® Java™ Development Environment includes RAPC, a command line compiler. RAPC compiles .java and .jar files into .cod files that you can run in the BlackBerry device simulator or load onto a BlackBerry device.

The rapc.exe file exists in the **bin** subdirectory of your BlackBerry JDE installation.

Note: `net_rim_api.jar` is required as an input file when you invoke RAPC. Use the command line argument with the `-import=` option to provide this .jar file to RAPC.

RAPC accepts the following command line options in the following order:

Option	Option format	Description
`java | class | jar`		The input files: • .java: A Java source program file that javac must compile. • .class: A Java .class file that javac has compiled. • .jar: An archive of files that the compilation set must include.
`jad`		An input file that contains application information. For example, it contains a list of attributes as specified by the MIDP specification.
`-class`	<*classname*>	The name of the class containing the application main entry point; without this option, RAPC uses the first `main(String[])` method it finds as the entry point.
`-codename`	=[<*path*>\[...]]<*filename*>	Specify the name and location of the output .cod file; typically the output .cod file uses the same name as the .jar file.
`-library`	=[<*path*>\[...]]<*filename*>	Specify the name and location of the output .cod file as a library.
`-import`	=<*file*>.jar[;...]	List dependent .jar files; for example list RIM APIs and other dependent libraries.
`-midlet`		Specify the .cod file as a MIDlet and generate a preverified .jar file.
`-deprecation`		The Java compiler ignores the value specified with the **-deprecation** option.
`-nowarn`		The Java compiler does not issue warnings for the value specified with the **-nowarn** option.
`-quiet`		Display only errors.
`-warning`		Generate warning messages.
`-verbose`		Display information about RAPC activity. RAPC stores this information in intermediate and temporary files in the BlackBerry device user's temporary folder. RAPC does not delete the temporary files.
`-wx`		Treat certain warnings as errors.

Option	Option format	Description
`-warnkey`	=0xNNNNNNNN[;...]	Generate a warning if you need to add a key to the .csl file.
`-workspace`	=<*filename*>	Add the <*filename*> to the .debug file for BlackBerry Integrated Development Environment browsing.
`filename_1.java [<additional .java files as required>]`		Specify the .java file name if you are compiling from .java files.
`JAR_filename.jar`		Specify the .jar file name if compiling from a .jar file.

Note: If you specify both the -**codename** and -**library** options, RAPC uses the -**library** option. For option values that start with an '=' symbol (for example: -**workspace**), the '-' before the option name is optional.

For example, the following command line instruction compiles the SampleApp.jar file into a .cod file of the same name:

```
rapc import=net_rim_api.jar codename=SampleApp\SampleAppDriver -midlet SampleApp.jad
Samples\SamplaApp.jar
```

B

Appendix: XML control entity attributes

Using XML control entity attributes

Using XML control entity attributes

Use the PAP DTD to specify the following attributes:

Goal description	XML control entity attributes	Example
Specify the equivalent of the REQUEST URI HTTP parameter for RIM push.	`X-Wap-Application-Id`	"/"
Specify a unique message ID. Additionally, use this control entity attribute to cancel or check the status of a message. Use a URL in combination with a value. For example, 123@blackberry.com.	`push-id`	123@wapforum.org
Specify the URL to which the result notification is sent.	`ppg-notify-requested-to`	http://wapforum:8080/ReceivePAPNotification
Specify the date and time by which to deliver the content to the BlackBerry® device. Content that has not been sent by this date and time is not delivered. Represent the date and time in UTC format: YYYY-MM-DDThh:mm:ssZ where • YYYY is a 4-digit year • MM is a 2-digit month • DD is a 2-digit day • hh is a 2-digit hour based on 24-hour timekeeping system • mm is a 2-digit minute • ss is a 2-digit second • Z indicates that the time is in UTC	`deliver-before-timestamp`	2004-01-20T22:35:00z
Specify the date and time after which content is delivered to the BlackBerry device. Content is not delivered before this date. Represent the date and time in UTC format.	`deliver-after-timestamp`	2004-01-20T21:35:00z
Specify the address of the BlackBerry device to which the push content is sent. The *destination* is the destination email address or PIN.	`address-value`	WAPPUSH=*destination*%3A*portI*/TYPE=USER@blackberry.com
Specify the delivery reliability mode of the content, transport-level, or application-level.	`delivery-method`	confirmed; unconfirmed

See the *Push Access Protocol (WAP-247-PAP-20010429-a)* specification at http://www.wmlclub.com for more information about writing server-side push applications using PAP. See the *PAP 2.0 DTD* for information about the WAP Push DTDs.

C

Appendix: .alx files

Elements in BlackBerry application .alx files

Elements in BlackBerry application .alx files

Element	Attributes	Description
application	id	The application element contains the elements for a single application. The application element can also contain additional nested application elements. Nesting lets you require that, when an application loads onto the BlackBerry® device, its prerequisite modules also load onto the BlackBerry device. The id attribute specifies a unique identifier for the application. To provide uniqueness, use an ID that includes your company domain in reverse order. For example, com.rim.samples.docs.helloworld.
copyright	—	The copyright element provides copyright information, which appears in the Application Loader.
description	—	The description element provides a brief description of the application, which appears in the Application Loader.

directory	—	The directory element provides the location of a set of files. The directory element is optional. If you do not specify a directory, the files must exist in the same location as the .alx file. The directory element specifies the directory relative to the location of the .alx file.
		Directory elements are cumulative within an application.
		For example:
		`<application id="com.abc.my.app">`
		`<directory>MyCodFiles</directory>`
		`<fileset Java="1.0">`
		`<files>`
		`a.cod //resolves to <.alx location>\MyCodFiles`
		`b.cod`
		`</files>`
		`</fileset>`
		`<directory>MyCodFiles</directory>`
		`<fileset Java="1.0">`
		`<files>`
		`c.cod //resolves to <.alx location>\MyCodFiles\MyCodFiles`
		`d.cod`
		`</files>`
		`</fileset>`
		`</application>`

files	—	The files element provides a list of one or more application .cod files, in a single directory, to load onto the BlackBerry device.

Element	Attributes	Description
fileset	Java radio langid Colour	The fileset element includes an optional directory element and one or more files elements. It specifies a set of .cod files, in a single directory, to load onto the BlackBerry device. To load files from more than one directory, include one or more fileset elements in the .alx file. The Java attribute specifies the minimum version of the BlackBerry JVM with which the .cod files are compatible. The current JVM is Version 1.0. The Java attribute is required. The radio attribute lets you load different applications or modules onto the BlackBerry device depending on the network type of the BlackBerry device. Possible values include Mobitex, DataTAC, GPRS, CDMA, and IDEN. The radio attribute is optional. The langid attribute lets you load different applications or modules depending on the language support that BlackBerry device users add to their BlackBerry devices. The value is a Win32 langid code; for example: 0x0009 (English), 0x0007 (German), 0x000a (Spanish), and 0x000c (French). The langid attribute is optional. The colour attribute lets you load different applications or modules for color or monochrome displays. The value is a Boolean; true means color display and false means monochrome.
hidden	—	The hidden element hides a package so that it does not appear to BlackBerry device users in the Application Loader. To hide a package, add the following line: <hidden>true</hidden>. Use this element in conjunction with the required element to load the application by default, or set the requires tag to load this package if another application exists. Only corporate system administrators should use the hidden tag. This tag is not intended for use by third-party application vendors. **Note**: The BlackBerry Desktop Software Version 3.6 and later supports this element.
language	langid	The language tag lets you override the text that appears in the Application Loader when the Application Loader runs in the language that the langid attribute specifies. To support multiple languages, specify multiple language tags. To specify the name, description, version, vendor, and copyright tags for each language, nest these tags in the language tag. If you do not nest a tag, the text appears in the default language. The langid attribute specifies the Win32 langid code for the language to which this information applies. For example, some Win32 langid codes are: 0x0009 (English), 0x0007 (German), 0x000a (Spanish), and 0x000c (French).
library	id	You can use the library element instead of the application element. It contains the elements for a single library module. You cannot nest modules. By default, a library module does not appear in the Application Loader. Typically, use the library element as the target of a <requires> element, so that when a particular application loads onto the BlackBerry device, a required library also loads onto the BlackBerry device. **Note**: The BlackBerry Desktop Software Version 3.6 and later supports this element.
loader	version	The loader element contains one or more application element. The version attribute specifies the version of the Application Loader.
name	—	The name element provides a descriptive name for the application, which appears in the Application Loader.
required	—	The required element lets you force an application to load. The Application Loader selects the application for installation, and the BlackBerry device user cannot change this selection. Add the following line: <required>true</required>. Only corporate system administrators should use the required tag. This tag should not be used by third-party application vendors. **Note**: The BlackBerry Desktop Software Version 3.5 and later supports this element.

Element	Attributes	Description
`requires`	`id`	The `requires` element is an optional element that specifies the `id` of a package on which this application depends. This element can appear more than once, if the application depends on more than one application. When an application loads onto the BlackBerry device, all packages that the `<requires>` tag specifies also load onto the BlackBerry device. **Note**: The BlackBerry Desktop Software Version 3.6 and later supports this element.
`vendor`	—	The `vendor` element provides the name of the company that created the application, which appears in the Application Loader.
`version`	—	The `version` element provides the version number of the application, which appears in the Application Loader. This version number is for information purposes only.

D

Appendix: BlackBerry application .jad files

Properties of BlackBerry application .jad files

Properties of BlackBerry application .jad files

Required RIM attribute	Description
RIM-COD-Creation-Time	creation time of the .cod file
RIM-COD-Module-Dependencies	list of modules that the .cod file requires
RIM-COD-Module-Name	name of the module that the .cod file contains
RIM-COD-SHA1	SHA1 hash of the .cod file
RIM-COD-Size	size (in bytes) of the .cod file
RIM-COD-URL	URL from which the .cod file can be loaded

Optional RIM attribute	Description
RIM-Library-Flags	reserved for use by Research In Motion (RIM)
RIM-MIDlet-Flags	reserved for use by RIM
RIM-MIDlet-NameResourceBundle	name of the resource bundle on which the application depends
RIM-MIDlet-Position	suggested position of the application icon on the Home screen **Note:** This position might not be the actual position of the application icon on the Home screen.

The BlackBerry® Integrated Development Environment lets you create a dual-purpose .jad file to support the downloading of MIDlets onto BlackBerry devices and other wireless devices. To do this, create a .jad file that contains both the `RIM-COD-URL` and `RIM-COD-Size` attributes and the `MIDlet-Jar-URL` and `MIDlet-Jar-Size` attributes. On BlackBerry devices, download the .cod files; on other wireless devices, download the .jar files.

Set .cod file dependencies

The application descriptor (.jad) file contains a `RIM-COD-Module-Dependencies` attribute that specifies the modules that the application requires, but are not provided with it. The `RIM-COD-Module-Dependencies` attribute lets a BlackBerry device user avoid downloading an application that will not run.

The `RIM-COD-Module-Dependencies` attribute takes a comma-separated module name list as a parameter. For example, an application that requires the RIM XML library might use the following entry in the application descriptor:

```
RIM-COD-Module-Dependencies: net_rim_cldc, net_rim_xml
```

Acronym list

A

AES
: Advanced Encryption Standard

API
: application programming interface

APN
: Access Point Name

ATR
: Answer To Reset

B

bpm
: beats per minute

C

CAC
: common access card

CDMA
: Code Division Multiple Access

D

DTD
: Document Type Definition

DTMF
: Dual Tone Multiple Frequency

G

GPRS
: General Packet Radio Service

GPS
: Global Positioning System

H

HTTP
: Hypertext Transfer Protocol

HTTPS
: Hypertext Transfer Protocol over Secure Sockets Layer

I

ID
: identification

L

LMM
: Low Memory Manager

M

MIDP
: Mobile Information Device Profile

P

PAP
: Push Access Protocol

PDE
: Position Determination Entity

PME
: Plazmic Media Engine

POST
: power-on self-test

R

RAM
: random access memory

S

S/MIME
: Secure Multipurpose Internet Mail Extensions

T

TCP
: Transmission Control Protocol

TCP/IP
: Transmission Control Protocol/Internet Protocol

Triple DES
: Triple Data Encryption Standard

U

UDP
: User Datagram Protocol

UI
: user interface

URI
: Uniform Resource Identifier

USB
: Universal Serial Bus

UTC
: Coordinate Universal Time

W

WAP
: Wireless Application Protocol

X

XML
: Extensible Markup Language

©2006 Research In Motion Limited
Published in Canada.